Efficient Non-uniform Fast Fourier Transform (NuFFT) Implementation for MRI Processing on FPGA

BY

EMANUELE PEZZOTTI
B.S, Politecnico di Torino, Turin, Italy, 2014

THESIS

Submitted as partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Chicago, 2017

Chicago, Illinois

Defense Committee:

Rashid Ansari, Chair and Advisor
Zhichun Zhu
Mariagrazia Graziano, Politecnico di Torino

ACKNOWLEDGMENTS

This thesis is the final part of my double degree project between University of Illinois at Chicago and Politecnico di Torino. I want, first of all, to thank both the institutions for this project and for the amazing time that I had here.

I would like to thank my advisor Prof. Rashid Ansari of the University of Illinois at Chicago for the opportunity of this research and for the time spent helping us in building the project. Similarly I would like to thank my advisor from Politecnico di Torino, Prof. Mariagrazia Graziano, for her availability and support even from a very long distance.

I would also like to thank Greg, for the experience he brought into the project and for the great support that he provided us during these months, and Umer, for the solid foundation we found when started working at the FPGA architecture. I would like to thank Intel-Altera for the economic support (grant 2016-03393) and for providing us with the latest FPGA in the market.

A big thank goes also to Alex for his patience and collaboration during the project. He was fundamental both from the pure practical research point of view, and from the support he provided me from emotionally during some difficult time.

I would like also to thank Lynn for managing the project so well and with such enthusiasm.

Finally a big thank goes also to my family, for the economical and emotional support they were able to give me, to Alessia for her patience and her figure always present during this year in Chicago,and to all my friends here and back in Italy, always available and supportive.

EP

CONTRIBUTION OF AUTHORS

Chapter 1 is a brief introduction to the problem addressed and the outline of the document. Chapter 2 is starting an analysis fo the MRI reconstruction from the physical background through the signal processing needed to tackle the problem. The interpolation kernel study has been addressed by Alex Iacobucci and it is reported in depth in his thesis. Chapter 3 is analysis I made for the density compensation step of the Non-uniform Fast Fourier Transform algorithm. Chapter 4 contains the proposed implementation (intended for a publication in a conference (1)) that has been developed in group. The actual OpenCL implementation is described more in depth in this document while Alex Iacobucci is focusing more on the software synchronization and the preprocessing described in the implementation. I made Figures 11-12-13 to describe the architecture for the research group. Fixed point and Non-scalable implementation are extensions I made for the research. Chapter 5 provides the results obtained for the proposed implementations (intended for a publication in a conference (1)), Figures 26 and 27 are intended for the publication and can be found in this thesis and in Alex Iacobucci's thesis (2) as part of the research (1). I made Figure 22 to emphasize the importance of the density compensation step, for this reason it can be found also on Alex Iacobucci document. Chapter 6 propose some further improvement to the architecture. The FFT modified kernels is part of the research, while the other two methodologies are extensions I made to the research. Chapter 7 provides a brief conclusion to the research presented in this document.

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

CHAPTER **PAGE**

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES (continued)

FIGURE **PAGE**

LIST OF ABBREVIATIONS

AMS	American Mathematical Society
CTAN	Comprehensive TEX Archive Network
TUG	TEX Users Group
UIC	University of Illinois at Chicago
UICTHESI	Thesis formatting system for use at UIC.
FPGA	Field Programmable Gate Array.
FFT	Fast Fourier Transform.
IFFT	Inverse Fast Fourier Transform.
NuFFT	Non Uniform Fast Fourier Transform.
MRI	Magnetic Resonance Imaging.
LE	Logic Elements.
FF	Flip Flop.
LUT	Lookup Table.
DSP	Digital Signal Processor.
RAM	Random Access Memory.
SSIM	Structural SIMilarity.
PSNR	Peak Signal to Noise Ratio.

SUMMARY

The use of MRI machines has recently witnessed tremendous growth in the medical field, largely due to its non-invasive acquisition of data about patients in a large variety of applications.

Over the years, a major challenge in MRI machines that scientists and researchers have been trying to address is improving the acquisition time. Historically MRI data has been sampled on Cartesian grids since such a grid guarantees ease in data processing due to the availability of fast algorithms to perform the required operation of an Inverse Discrete Fourier Transform (IDFT). However this advantage comes with the cost of long acquisition time over Cartesian grids which makes it tedious for patient inside an MRI machine.

To overcome this, the scientific community sought to reduce the acquisition time by sampling on more complex trajectories. The saving in acquisition, however, increases processing complexity in performing the IDFT on non-Cartesian grids.

A fast algorithm that includes all the necessary steps for reconstructing the IDFT image from the non-uniform grid is called the Non-uniform Fast Fourier Transform (NuFFT). The overall aim of this thesis is to devise and implement an efficient NuFFT algorithm using FPGAs and OpenCL and to to investigate the functionality and performance of the design.

Specifically, this document focuses on possible techniques to handle the density compensation step, proposing different approaches and comparing them in terms of performance, accuracy of reconstruction and hardware resources utilization. An efficient, low power, and scalable architecture suitable for the most commonly used MRI image sizes is presented, achieving a

marked improvement with respect to previously published FPGA and CPU implementations. Furthermore, a fixed point implementation of the designed architecture is proposed, achieving a considerable improvement in term of resource utilization which may be useful for future 3D extensions of the same architecture.

Finally, a new technique exploiting the symmetric property of the k-space is presented, potentially achieving large improvements in performance and memory utilization at a minor cost in the accuracy of the image reconstruction.

CHAPTER 1

INTRODUCTION

Magnetic Resonance Imaging (MRI) is one of the most used methods for imaging the body, due to its non invasive nature and to the excellent tissue contrast that can be obtained. However, MRI requires long scan times to obtain an image making the process long and tedious for patients inside the machine.

Over the years, several works have been proposed in order to reduce the acquisition time of an MRI machine, both from a theoretical and practical point of view. Non-Cartesian sampling trajectories allow accurate and efficient image reconstruction, increasing the signal processing phase in order to properly reconstruct the image. The purpose of this work is to create an efficient FPGA accelerator to elaborate data from an MRI machine in a reasonable time with acceptable accuracy. This project was part of a research group funded by Altera, and part of the design, like the study of the interpolation kernel, as well as some hardware and software reordering techniques applied in the FPGA, is also presented in Iacobucci et al. [2].

1.1 Thesis Outline

Chapter two provides a brief introduction to MRI, from the physical working background to the acquisition of the signal stimulated using two different magnetic fields, to the filling of the frequency space (called k-space) using different trajectories. Then the Fourier Transform is introduced to transform the signal from the frequency domain to the actual image domain.

The concept of Non-Uniform Fast Fourier Transform and how it is performed is described, introducing all the steps that the accelerator has to perform in order to properly reconstruct an MRI image.

In chapter three, the density compensation step (probably the most effective in terms of accuracy of the image reconstruction) is taken into account. Several methodologies to tackle this step are presented. The Voronoi diagram is also introduced, allowing one to describe a density estimation based on this mathematical instrument.

In chapter four the actual implementation using OpenCL on FPGA is described. First of all, OpenCL is introduced in order to describe the programming scheme ideal for this programming language. Then the proposed implementation is presented with a description of every single component (hardware and software) in the architecture. The main purpose of this architecture is to make it scalable and efficient at the same time. The focus then shifts to performance leading to a proposed non-scalable approach, as well as a Fixed Point implementation so as to reduce the hardware resource utilization.

In chapter five the results of the different implementations are reported. A comparison in terms of performance, hardware resources utilization, and accuracy is performed for different ways to compute the density compensation. The proposed architecture results are shown, as well as the comparison with the non scalable implementation. Finally fixed point implementation is compared with the floating point in terms of accuracy, performance and hardware resource utilization.

In chapter six additional approaches are presented. Some of them require extra hardware to improve the kernel execution time, while others exploit theoretical properties of the k-space to simplify and reduce the number of points taken into account.

Finally chapter eight summarizes the work commenting on the results obtained.

CHAPTER 2

MRI RECONSTRUCTION THEORY

2.1 Physical background

In order to understand how an MRI machine works a simplified description of the physical principles besides the Magnetic Resonance theory are provided.

Matters is comprised of atoms and each atom includes 3 main particles: electrons, protons and neutrons. Electrons continuously move around the nucleus of the atom where protons and neutrons reside.

Elements in nature can be characterized by their atomic number (number of protons inside an atom), and by the atomic weight (given by the number of protons and neutrons).

Elements with an even atomic number and even atomic weight do not have *spin* which is fundamentally related to the Magnetic Resonance [3]. Protons inside the nucleus are repeatedly rotating around an an axis creating their own magnetic field that is oriented along the axis of rotation. A proton placed within a magnetic field B_0 starts rotating around the axis of the magnetic field direction. This phenomenon is called Magnetic Resonance.

Actually, two different states are possible when protons are within an external magnetic field. They can, in fact, rotate in a parallel or anti-parallel way with respect to the magnetic field direction. The rotation frequency around the axis is determined by the Larmor Equation 2.1:

$$f_0 = \gamma \cdot B_0 \tag{2.1}$$

where:

- γ is a constant called GyroMagnetic Ratio, which is a characteristic of every proton.
- B_0 is the applied Magnetic Field in Tesla.

When protons are not within a strong external magnetic field they align almost randomly in every direction so, if we sum up all the magnetization vector, a zero total magnetization is obtained. Once a strong external magnetic field is applied, they will align along that direction and the magnetization will be a value greater than 0. The greater is the intensity of the external magnetic field, the higher will be the value of the magnetization since the number of protons rotating in the anti-parallel direction will be lower.

Now in order to produce an MRI signal, another magnetic field B_1 needs to be applied so as to produce a dynamic magnetic vector. The B_1 force applied needs to rotate at the same speed of the continuously rotating spins (f_0). Under this condition it is possible to push the rotating spins from the longitudinal plane (z) to the transverse one (xy).

Once the magnetization M is tilted in the xy plane, we can measure the intensity of the induced current in a perpendicular coil exploiting Faraday's principle.

2.2 MRI Encoding and Gradients

Thanks to the interaction between protons and magnetic fields and thanks to the Faraday's principle it is possible to measure an MRI signal using a coil. It is also possible to change the total magnetization and produce a signal, but it is not possible to locate the origin of the signal.

All the protons of the same region of interest are precessing at frequency f_0, which is dependent on the main static field B_0. A change in the strength of the field with respect to the

location will result in a precession of each proton with a different frequency. So, observing the measured MR frequency component it is possible to spatially locate the protons. Spatial encoding is the name given to the frequency labeling accomplished by changing gradients (magnetic fields used in the acquisition process). To locate then in a plane we need at least 2 coordinates, so spatial encoding in two directions is needed [4].

Frequency Encoding Gradients

Frequency encoding is accomplished through the use of magnetic fields called gradients. Gradients distort the main magnetic fields in a predictable way, causing a change in the precession frequency as a function of the position.

The effective magnetic field $B(x)$ at any point (x) along the horizontal axis can be computed by:

$$B(x) = B_0 + xG_f \tag{2.2}$$

where B_0 is the main magnetic field, and G_f is the frequency-encoding gradient. From the Equation 2.1 the resonant frequency is varying linearly along the frequency-encoding axis:

$$f(x) = \gamma B(x) = \gamma B_0 + \gamma x \cdot G_f = f_0 + f_G(x) \tag{2.3}$$

where f_0 is the main Larmor frequency. Figure 1 shows how the frequency encoding works along the horizontal axis.

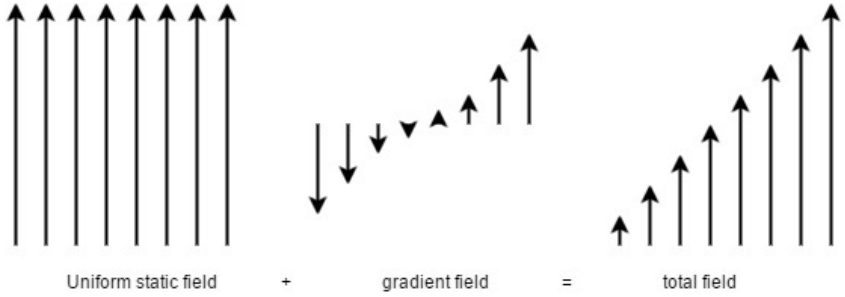


Figure 1: Frequency Encoding

Phase Encoding Gradients

Phase encoding is more difficult than frequency encoding. The phase encoding gradients add another signal to the contribution due to the frequency gradient.

The measurement is performed in two steps: in the first one the phase encoding is in phase with the frequency encoding, while in the second one the gradient is applied and the phase is slightly changing on the vertical axis.

The measured signal will always be the sum of the two contributions, but it is very important to extract the phase from the second contribution to spatially locate the point. This is the reason why two steps are needed. With two measurements it is always possible to extract both the contributions and so the phase.

Combining these two techniques it is possible to spatially locate the informations. In fact, all the elements in a column will have the same frequency but a different phase, and all the elements in a row will have the same phase but a different frequency.

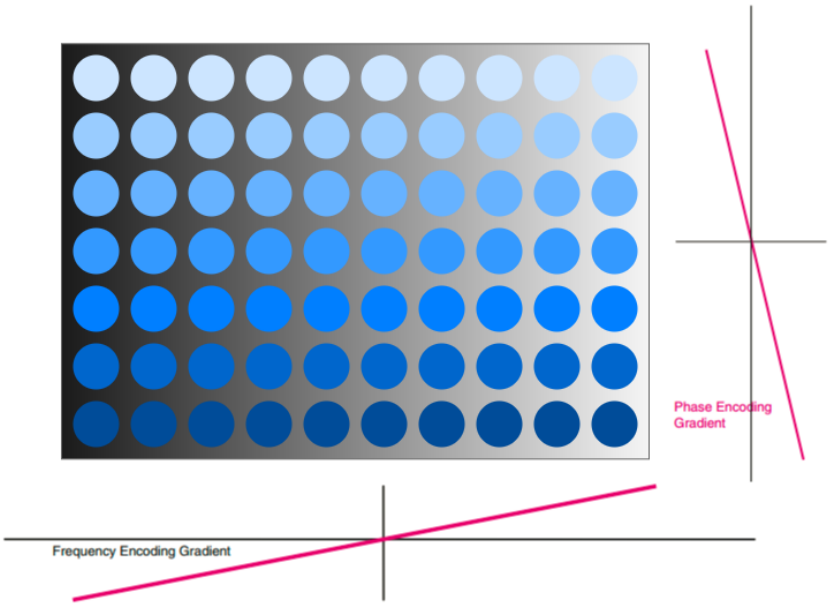


Figure 2: Frequency Encoding (background gradient) and Phase Encoding (circles gradient). Image adapted from [3].

Figure 2 shows both the frequency and the phase encoding. The frequency encoding is the background gradient, while phase one is the change in color between the circles.

2.3 K-space

The complex raw data set obtained by collecting all the elements in phase and frequency encoding is called k-space. K-space is often referred to as the temporary image space, in which data from MRI signals are stored during an acquisition.

K-space is a spatial frequency domain. Its 2-D Fourier Transform will lead to the final reconstructed image. For that reason, the points (k_x, k_y) in k-space do not correspond with individual pixel (x, y) in the image. Each point in k-space contains informations about every pixel of the final image.

2.4 K-space Trajectories

The process of acquiring data using an MRI machine is also called k-space filling, and it can be done in many different ways depending on how we activate the phase and frequency encoding [5]. Every single method offers benefits and disadvantages:

- **Linear or Cartesian**: it is accomplished sensing line by line. This is the easiest way to collect data in the frequency domain and it does not require any processing since it is possible to apply directly the 2D Inverse Fast Fourier Transform in order to produce the final reconstructed image.
- **Centric**: it is accomplished sensing first the amplitudes in the center of the k-space and then going to the periphery. It is useful when a contrast MRI is required. In fact, low frequency signals are collected quickly allowing a visualization of the contrast media in the minimum amount of time possible.

- **Spiral**: it is accomplished by sensing the k-space in a spiral fashion starting from the center and going towards the periphery. As with the Centric one, this is beneficial in contrast enhanced MRI imaging. Furthermore, it is more efficient than the Cartesian one since it achieves a greater coverage of the entire k-space in less time, and does not collect data in the corners of the k-space that don't necessarily contribute to the final image.
- **Radial**: it is accomplished by sampling the k-space with radial spokes that pass through the center of the k-space. There are several potential advantages in the use of the radial trajectory, such as the shorter minimum TE or the no phase-encoding requirement.

A representation of the 4 trajectories can be found in Figure 3.

It is worth noting that the scan time can be reduced changing the way the k-space is filled. Since the k-space is obtained changing the polarity of the gradient applied in the two directions (frequency and phase), the top half part of the k-space is a mirror image of the bottom half with opposite polarity. It is possible to state the same for left and right part.

Two different techniques exploit these characteristics of the k-space to make the scan time shorter:

- **Half Fourier**: it is accomplished filling half of the k-space along the phase direction. The other half can estimated from the acquired data.
- **Partial Echo**: similar to the Half Fourier it is accomplished filling half of the k-space in the frequency encoding direction.

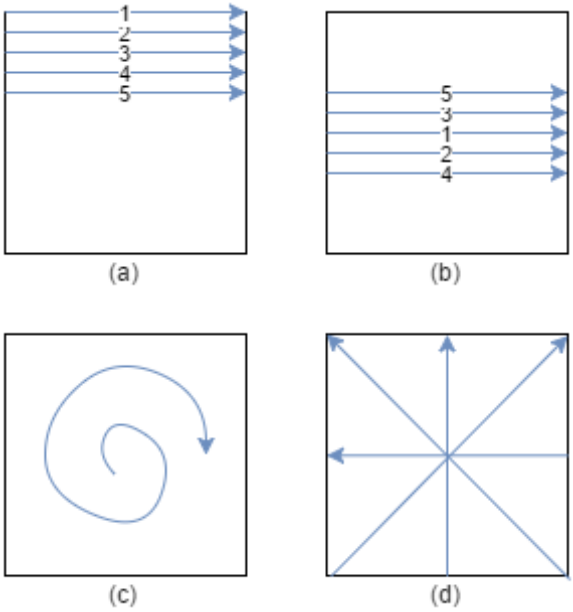


Figure 3: Representation of the 4 most used trajectories in MRI. (a) is Cartesian, (b) is Centric, (c) is Spiral, (d) is Radial

2.5 Fourier Transform

The Fourier Transform is a mathematical operation that converts a signal in time domain $s(t)$ into a signal in the frequency domain $S(f)$ according to the following formula.

$$S(f) = \int_{-\infty}^{+\infty} s(t) \cdot e^{-2\pi f t} dt \tag{2.4}$$

Where:

- t and f are real variables.
- $s(t)$ and $S(f)$ may be complex.

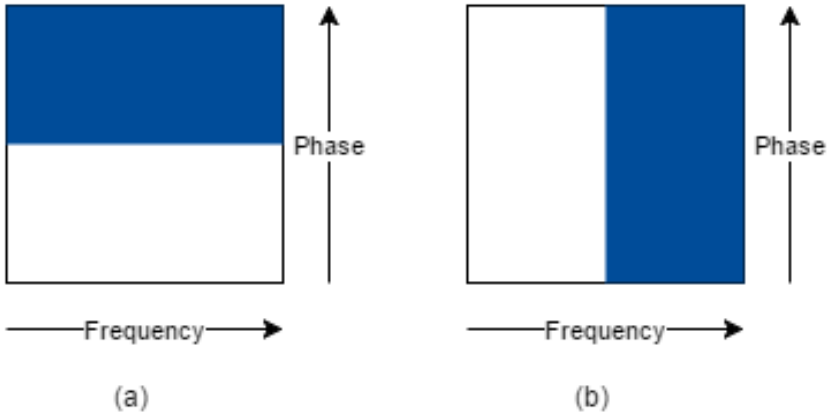


Figure 4: In the Half Fourier (a) just half of the k-space is filled in phase encoding direction, while in Partial Echo (b) half of the k-space is filled in the frequency encoding direction

- $e^{-2\pi ft}$ is often referred to as a *transform kernel*.

Equation 2.4 shows how a signal in the time domain can be transformed into oscillatory functions in the frequency domain. Likewise knowing a frequency representation of a signal, as is the case in the MRI field, it is possible to retrieve $s(t)$ by applying the Inverse Fourier Transform:

$$s(t) = \int_{-\infty}^{+\infty} S(f) \cdot e^{2\pi ft} df \tag{2.5}$$

In the digital world, the considered signal are never continuous but are usually known at N instants separated by sample times T. The counterpart of the Fourier Transform for finite-extent discrete-time signals is called Discrete Fourier Transform. The Discrete Fourier Transform and its inverse are defined by the following formulas:

$$S(f) = \sum_{t=0}^{N-1} f(t) \cdot e^{-2\pi ft} \tag{2.6}$$

$$s(t) = \sum_{f=0}^{N-1} S(f) \cdot e^{2\pi ft} \tag{2.7}$$

The concept of the Fourier Transform can be extended and generalized to higher dimensions. In case of an MRI we are mostly interested in 2D or 3D cases. Since the purpose of this thesis is to describe an architecture to elaborate and process MRI images, the focus will be on 2 dimensional discrete-space signals. The 2 dimensional Discrete Fourier Transform and its inverse is defined by the following formulas:

$$S(k,l) = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} s(m,n) \cdot e^{-j2\pi(\frac{k}{M}m + \frac{l}{N}n)} \tag{2.8}$$

$$s(m,n) = \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} S(k,l) \cdot e^{-j2\pi(\frac{k}{M}m + \frac{l}{N}n)} \tag{2.9}$$

Where:

- k, l are the coordinates in the frequency domain
- m, n are the coordinates in the image domain
- M, N are the numbers of uniformly sampled data in the two directions

It is worth noticing that, as Pratt says in [6], "because the transform kernels are separable and symmetric, the 2D transform can be computed as sequential row and column 1D transforms".

This approach has been used in order to compute the Inverse Architecture in the OpenCL FPGA implementation.

2.6 Non Uniform Fast Fourier Transform

In MRI applications and many more in the signal processing area, data is not always uniformly sampled. Standard Discrete Fourier Transform is defined for regularly sampled data of finite-extent signals, so, non-uniform trajectories during the k-space filling preclude the use of this mathematical instrument for the image reconstruction.

Non-Uniform Discrete Fourier Transform is a generalization of the DFT for data that are not uniformly sampled.

Consider a set S of size M consisting of non-equispaced source samples s_i, $i = 1, ..., M$. Let the reconstructed image f be of size $G \times G$, and let m, n be the matrix coordinates in f with an offset of $G/2$ ($m, n = -G/2, ..., G/2 - 1$). NuDFT can be formally defined as in [7]:

$$f(m,n) = \sum_{i=1}^{M} s_i \cdot e^{-2\pi i(s_i x m + s_i y n)} \tag{2.10}$$

with s_i being the complex value of the source point, and $s_i x$, $s_i y$ being the k-space coordinates of point s.

The arithmetic operations required for a computation of the NDFT are $O(MG^2)$. With an approximation scheme the NuFFT allows the computation in $O(M + G^2 log G)$ complexity [8]. Figure 5 shows the main steps of the algorithm.

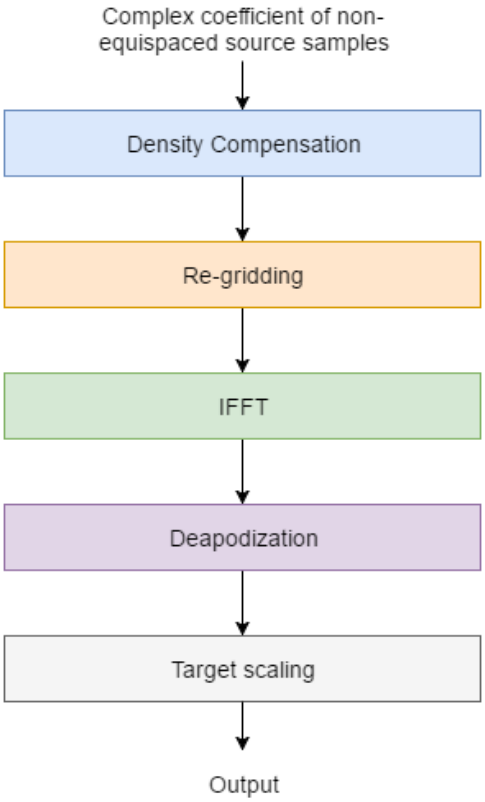


Figure 5: Non Uniform Fast Fourier Transform steps

Each step of the algorithm will be explained in this chapter. Different density compensation techniques will be analyzed in the next chapter while some others will be just briefly described since they will be explained in greater detail in [2].

Density Compensation

The use of non-Cartesian trajectories introduce different densities in the k-space sampling. Some regions of the k-space can have a high concentration of samples while others have much lower. Each and every kind of trajectory will introduce different patterns in the density of

samples in the k-space. Some of them will lead to a higher concentration in the center of the k-space (radial, spiral) while others will lead to more points at the edges (lissajous).

The use of these different trajectories in the MRI machines are briefly explained next. The central frequencies (lower frequencies) contain the most important information in the image reconstruction so the data acquisition must be more precise, thus at those frequencies denser. Sampling points far form the k-space origin will lead to acquisitions of higher frequency content (edges in a reconstructed image) that does not carry the same amount of fundamental information.

It is critical to consider the different sampling densities so to normalize the obtained values and to properly reconstruct the image.

Re-gridding

The gridding is the fundamental step in the actual reconstruction [9].
Data samples lie along a trajectory in the k-space. For each target point computation a convolution is performed with an interpolation kernel and the result is sampled and accumulated into a Cartesian grid.
Mathematically speaking, as shown by Pauly in [10], the gridding can be described in the following way. The non-Cartesian sampling function (trajectory) $S(k_x, k_y)$ is:

$$S(k_x, k_y) = \sum_i \delta(k_x - k_{x,i}, k_y - k_{y,i}) \tag{2.11}$$

The sampled data is given by product of the value of the k-space at that point and the trajectory function described above $M(k_x, k_y) \cdot S(k_x, k_y)$. This is convolved with the chosen interpolation kernel $C(k_x, k_y)$ and then sampled on a Cartesian grid [10]:

$$\hat{M}(k_x, k_y) = [(M(k_x, k_y) \cdot S(k_x, k_y)) * C(k_x, k_y)] \cdot \text{Ш}\left(\frac{k_x}{\Delta k_x}, \frac{k_y}{\Delta k_y}\right) \tag{2.12}$$

There are some fundamental parameters that have to be chosen in this step in order to properly reconstruct an image.

1. Type of convolution kernel $C(k_x, k_y)$
2. Density of the reconstruction grid S.
3. Size of the interpolation window W.
4. Oversampling factor α.

The choice of these parameters can deeply affect both performance and quality of the results of the reconstructed image. There are several convolution kernels that can be used to interpolate the non-uniform sampled points. The most well-known ones are described and compared in several papers in the literature and they are: Gaussian, 2 terms cosine, 3 terms cosine, Kaiser-Bessel. Kaiser-Bessel has been proved to be the most efficient and the one that guarantees the lowest aliasing in the reconstructed image [2]. For that reason, it has been chosen for analysis and implementation in the proposed FPGA architecture.

The density of the Cartesian grid is another parameter that we are free to choose since we are reconstructing on that grid. This choice is going to affect the aliasing generated from the

adjacent replica lobes and on the apodization effect that the kernel produces. It is important to carefully choose this parameter.

The size of the interpolation window is the last important parameter that has to be chosen. This affects performance and quality of the reconstructed image. The bigger the size of the window the worse the performance is, since more computations are required for each source point. For example, if the size of the interpolation window is 4, each source point will affect and update 16 target points. If 5, then 25 target points will be modified and so on. On the contrary, increasing the size of the interpolation window is not necessarily going to improve the quality of the image. The number that provided the best performance with a satisfactory quality of the reconstruction image was 4.

The oversampling factor α defines the minimum distance between two points in the uniformly sampled target space. The smaller the distance the smaller will be the error caused in the reconstruction by the aliasing replica. The oversampling factor will significantly affect the use of local memory in the FPGA, and consequently the performance.

Figure 6 and Figure 7 show the result of the regridding step. Non uniformely sampled data points are convolved with an interpolation function creating a surface. This surface is, then, uniformly sampled.

2D Inverse Fast Fourier Transform

The algorithm used to implement the 2D Inverse FFT is the Cooley-Tukey. It is the most commonly used FFT algorithm. It exploits one of the characteristics of the FFT in order to

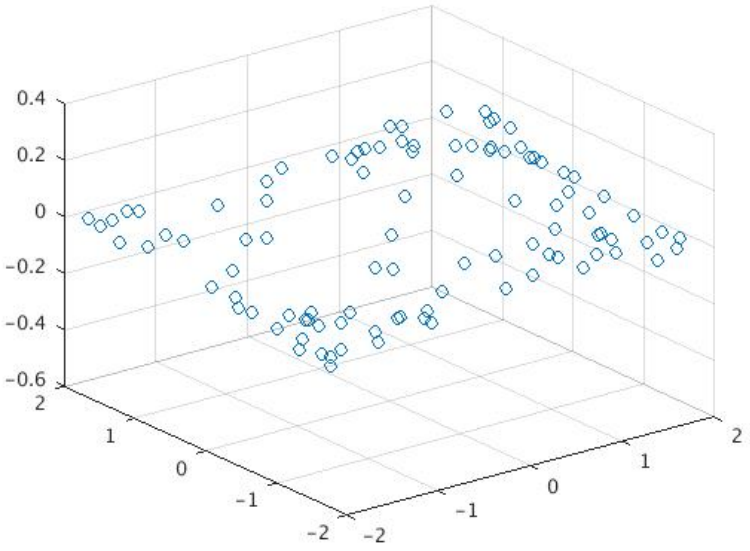


Figure 6: Non uniformely sampled data

make it easier and faster to apply. It consist of applying the 1 dimensional FFT on each row and then on each column of the 2 dimensional space. Mathematically speaking [10]:

$$\hat{m}(x,y) = [(m(x,y) * s(x,y)) \cdot c(x,y)] * \text{Ш}\left(\frac{x}{FOV_x}, \frac{y}{FOV_y}\right) \tag{2.13}$$

Deapodization

An error is also introduced by the inverse transform of the adopted kernel (called apodization function). This error can be easily compensated by dividing the image with the ideal apodization function. Practically speaking, it is not convenient to apply the deapodization to the entire

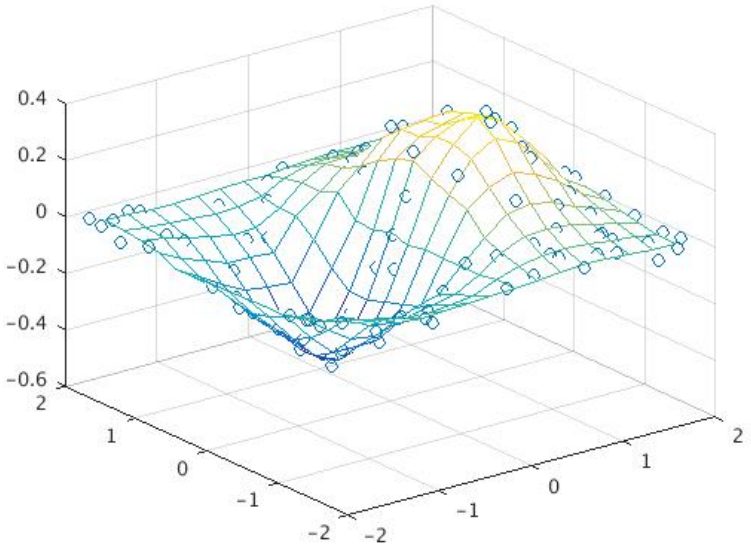


Figure 7: Interpolated Non Uniformely Sampled Data

image, because it can accentuate some artifacts at the edge of the FOV. The point of interest is anyway just the center of the image [9]. Mathematically speaking [10]:

$$\hat{m}(x,y) = \frac{1}{c(x,y)+a}[(m(x,y) * s(x,y)) \cdot c(x,y)] * \text{Ш}\left(\frac{x}{FOV_x}, \frac{y}{FOV_y}\right) \tag{2.14}$$

CHAPTER 3

DENSITY COMPENSATION

As stated previously, assigning the same weight to each k-space point when using Non-Cartesian trajectories with nonuniform density will lead to an inaccurate reconstruction [9].

The non uniform k-space density is the result of the different distance between portions of the same trajectory (different interleaves of the same spiral or radial acquisition) and from varying the velocity of the scan acquisition along the trajectory (usually the velocity is slow when passing along the center of the k-space, while it increases going farther from it).

If the regridding algorithm is used to reconstruct an MRI image this step is important to get an accurate result. Different approaches can be adopted to compute the density of the non uniform acquisition. Two macro-groups can be defined:

- Post-compensation: the compensation is applied after the regridding when the k-space is uniformly sampled. Mathematically speaking [10]:

$$\hat{M}(k_x, k_y) = \frac{1}{\rho'(k_x, k_y)}[(M(k_x, k_y) \cdot S(k_x, k_y)) * C(k_x, k_y)] \cdot \text{Ш}\left(\frac{k_x}{\Delta k_x}, \frac{k_y}{\Delta k_y}\right) \tag{3.1}$$

- Pre-compensation: the compensation is applied directly to the raw data from the MRI. It usually requires an analytic knowledge of the sampling trajectory used or some numeric computations. Mathematically speaking [10]:

$$\hat{M}(k_x, k_y) = [(\frac{M(k_x, k_y)}{\rho(k_x, k_y)} \cdot S(k_x, k_y)) * C(k_x, k_y)] \cdot \text{Ш}\left(\frac{k_x}{\Delta k_x}, \frac{k_y}{\Delta k_y}\right) \tag{3.2}$$

3.1 Post-compensation

This technique consists of keeping track of the quantity of the "energy" accumulated in each grid point during the regridding, and then performing a division by this energy after completing the convolution. It is easy to implement this in parallel to the regridding step. It can be view as regridding itself but considering the input data 1 instead of the complex value sampled by the MRI machine.

In this way, for each grid point the value of the "energy" given by the interpolation kernel it is stored and accumulated. To avoid division by 0 (in case a grid point is never affected) a constant small value is given to all the grid points. The value should be at least one order of magnitude smaller than the smallest value present in the kernel interpolation window.

The major advantage of this technique is that can be computed on the fly during the regridding. This means that we do not necessarily need to know the sampling trajectory of the MRI machine, or to perform preprocessing on the raw data.

However, it can be used only when the k-space sampling density varies slowly in space. It,

furthermore, interferes with the image deapodization step, and it does not work properly for commonly used k-space trajectories, such as spiral and radial.

3.2 Pre-compensation

As already stated, this technique refers to performing the density compensation prior to the convolution. It means that the weights for raw data needs to be assigned by the Density Compensation Function (DCF).

3.2.1 Analytic Pre-compensations

There are numerous ways to compute the DCF. Depending on the trajectory used it is easy to apply some geometric arguments to calcualte the DCF. For radial scans, for instance, the DCF is called Rho filter. The Rho filter function value is proportional to the distance from the center of k-space multiplied by the gradient magnitude. Similarly, for other trajectories it is possible to find some analytic funcitons that represent exactly the DCF needed to compensate for the non-uniform sampling trajectory.

Meyer and Nishimura in [11] proposed an analytic solution for spiral trajectories that consist in the multiplication of the raw source points by $|g(t) \cdot [sin(arg\{g(t)\} - arg\{k(t)\})]|$, where $g(t)$ is a function proportional to the k-space velocity acquisition.

The first term takes into account the the different k-space velocity, while the second one the higher density of the spiral in proximity of the origin.

In the literature different solutions have been proposed for an analytic computation of the sampling density for spiral trajectories [12] [13].

3.2.2 Numerical Pre-compensations

Analytic pre-compensations usually are not computationally intensive, and that is the reason why several approaches have been studied for the most commonly used trajectories.

However, for more general trajectories, it may not be possible to generate a DCF analytically. Numerical approaches can be directed at specific or more general trajectories and applied to all the possible k-space filling. For spiral trajectories different numerical techniques have been developed in [14] and in [15]. Since the purpose of this document is to create an architecture working for different kind of MRI machines, a general method based has been studied and analyzed. It consists of computing the Voronoi diagram of the sample distribution.

3.2.2.1 Voronoi Density Compensation

In order to fully understand how this method works it is important to define the Voronoi diagram [16].

It is assumed that a finite number n of points in an Euclidan plane is given, and that n is a finite number greater than 2. Let us label the points $p_1, p_2, ..., p_n$ with coordinates in a Cartesian plane $(x_{11}, x_{12}), ..., x_{n1}, x_{n2})$ and assume that such points have different coordinates in the plane. Let p be a point in the plane with coordinates (x_1, x_2), then the Euclidean distance between p and p_i is given by:

$$d(p, p_i) = \sqrt{(x_1 - x_{i1})^2 + (x_2 - x_{i2})^2} \tag{3.3}$$

If p_i is the nearest point to p, then p is assigned to $V(p_i)$, Voronoi region associated with p_i. Therefore, the region given by

$$V(p_i) = \{(x_1, x_2) | d(p, p_i) \leq d(p, p_j), for j \neq i, j \in I_n\} \tag{3.4}$$

is called Voronoi polygon associated with p_i. The Voronoi diagram is the set of the Voronoi polygon associated with each point in the plane:

$$Voronoi(P) = \{V(p_1), V(p_2), ..., V(p_n)\} \tag{3.5}$$

An actual example of the Voronoi diagram is Figure 8.

As can be seen, each point has a Voronoi polygon associated to it. The area of each Voronoi polygon is inversely proportional to the density. Points very close to each other (dense regions in the plane) will have a very small Voronoi polygon associated with them, while points in the outer space that are farther from each other will have bigger Voronoi polygons.

By applying the Voronoi diagram to MRI reconstruction, it is so possible to have a parameter proportional to the density for every sample in the raw data acquisition. This approach can be applied to all the trajectories, from the most commonly used to random ones, as long as there is no double sampling of a given point. One of the pre-requisites of the Voronoi diagram is, in fact, the uniqueness of the points in the plane.

Mathematically speaking the Voronoi diagram computation is more intensive than an analytical approach, but, since Voronoi diagram is a widely studies concept in many fields several

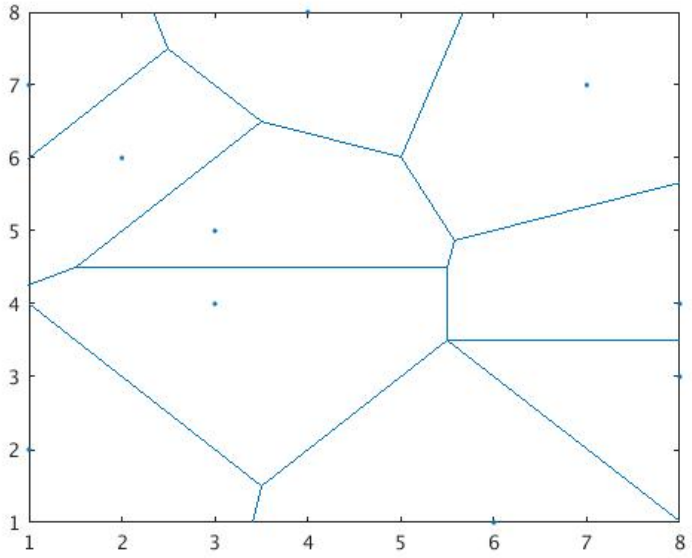


Figure 8: Example of the Voronoi diagram for a random of points in a space. Generated using MATLAB software

solutions to help and simplify the mathematical process have been proposed. On of the issue related to the Voronoi diagram applied as density compensation in MRI reconstruction is that the outer points are unbounded, thus the area of the Voronoi polygon associated to them is, theoretically speaking, infinite.

One of the first approaches to solve this issue, the one proposed in [17], is to determine the convex hull for the sampling points, and to remove the points belonging to the outer convex hull. This solution is convenient and does not affect the accuracy of the image reconstruction since it will remove just few points far from the center of the k-space. A similar approach is to identify the points in the outer convex hull and to assign to them the latest finite value

computed, or to the greatest one. These solutions are, however, increasing the complexity of the Voronoi diagram calculations, making this approach impractical for real-time systems. For the purpose of the architecture, it is possible to consider the acquisition trajectory of the MRI machine not changing over different acquisitions. For this reason it is possible to compute the Voronoi diagram and the density compensation once off-line before the actual computation.

However, in order to simplify the Voronoi diagram computation, the solution proposed in [18] it is used. It consists of the computation of the Voronoi diagram for the points close to the center of the k-space, and to assign the rest of the points areas equal to that of the last cell computed. Thanks to that approach, the number of points processed by the Voronoi function is drastically reduced. In terms of accuracy and performance of the density compensation, this approach leads to good results thanks to the fact that the points close to the center of the Fourier space carry most of the fundamental information of the image that needs to be reconstructed.

CHAPTER 4

FPGA IMPLEMENTATION

In this chapter all the different approaches adopted in order to fulfill the purpose of an MRI accelerator on FPGA using OpenCL will be described and analyzed. Then the proposed architecture for the FPGA 2017 conference will be explained in detail.

4.1 FPGA

An FPGA is "a large-scale integrated circuit that can be programmed after it is manufactured rather than being limited to a predetermined, unchangeable hardware function"[19]. FPGA are composed by small blocks of programmable logic, that contains registers, low level configurable elements arranged in a grid connected through a complex interconnection system. Logic elements are, usually, Look-up Tables combined with multiplexers and clock enables, and they can also contain higher-lever arithmetic blocks like multipliers and floating point blocks.

Recently, FPGAs have spread their role into computing units for processors and more complex systems. The great advantage of reconfigurability allows to create a platform highly efficient for different applications without the requirement of an actual change in the system. Unlike GPUs which are commonly used in order to help the main processor of a machine to perform particular operations, FPGAs are deeply linked with the hardware structure that a designer wants to infer. They do not exploit the programming parallelism, but they exploit the actual hardware that can be pipelined and replicated.

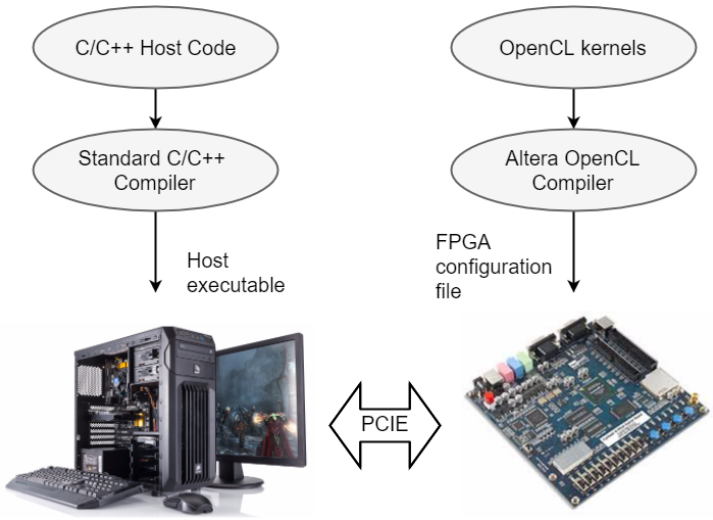


Figure 9: OpenCL FPGA working architecture

4.2 OpenCL

The very first point that needs to be addressed is the difference between programming an FPGA using OpenCL with respect to the standard HDL used until now. OpenCL is a framework for writing programs and applications across numerous heterogeneous platforms (CPUs, GPUs, DSPs, FPGAs). It has a specific view of the computing system, consisting of a number of *compute devices* attached to a *host* processor. All the functions executed on the OpenCL device are called kernels. Kernels are launched by the host processor through an application programming interface (API) that also manages the device memory, which is separate from the host device [20] [21].

Memory usage and management is fundamental to building an efficient accelerator. OpenCL defines a 4-level memory hierarchy for the compute device:

- **Global Memory**: is shared by all the processing elements and it should be used only when necessery since it has an high access latency.
- **Read-Only Memory**: it is small, with low latency, but it can be written only by the host device and not by the computing devices.
- **Local Memory**: it is small and it can be managed internally by the computing device and it has a reduced access latency.
- **Registers**: fastest memory available on the computing device. It is of limited size though.

There are several advantages that come with the use of OpenCL instead of the standard HDL based design. Traditional design consists in creating datapaths and state machines to control these datapaths, managing directly the hardware resources available and handling the timing constraints imposed by the external interfaces. The OpenCL compiler does all of that automatically, making the FPGA design similar to software development. This approach makes it easier to migrate from different typologies of FPGAs, leaving to the compiler the hardware management.

It, furthermore, reduces the development time making it easier to reduce and meet the time to market required for a product. However, in order to fully exploit the OpenCL compiler potential, some guidelines and precautions have to be followed while describing the functions in the kernels. The most relevant, identified during the time spent describing the architecture are the following:

- Loops should have a fixed number of iterations. This is due to the fact the OpenCL compiler, in order to increase the throughput of the entire design, tends to pipeline everything. In order to fully pipeline the exact number of iterations in a loop must be fixed.
- Simpler code translates into better hardware. This is due to the fact that the OpenCL compiler translates the C code for the kernel description into Verilog. The simpler the C code is, the easier it is for the compiler to translate into better hardware the behavior described in the kernel.
- Limited number of access to global memory. This is due to the fact that the global memory is slow, and represents, most of the time, the bottleneck. However, sometimes it may happen that global memory have to be accessed several times. In this case it is necessary to perform it in bursts.

In order to obtain good results using the Altera OpenCL compiler it is necessary to carefully follow these steps.

4.3 First Approach

This first approach has been developed without the use of a workstation. It is still tied to the HDL way to describe an architecture (the one proposed by Cheema et al. in [22]), and it led to poor performance. It is still important to mention that, since it was a fundamental step in the understanding of the actual use of OpenCL. The overall design can be found in Figure 10. Due to its complexity, a brief description of how the kernels are working will follow:

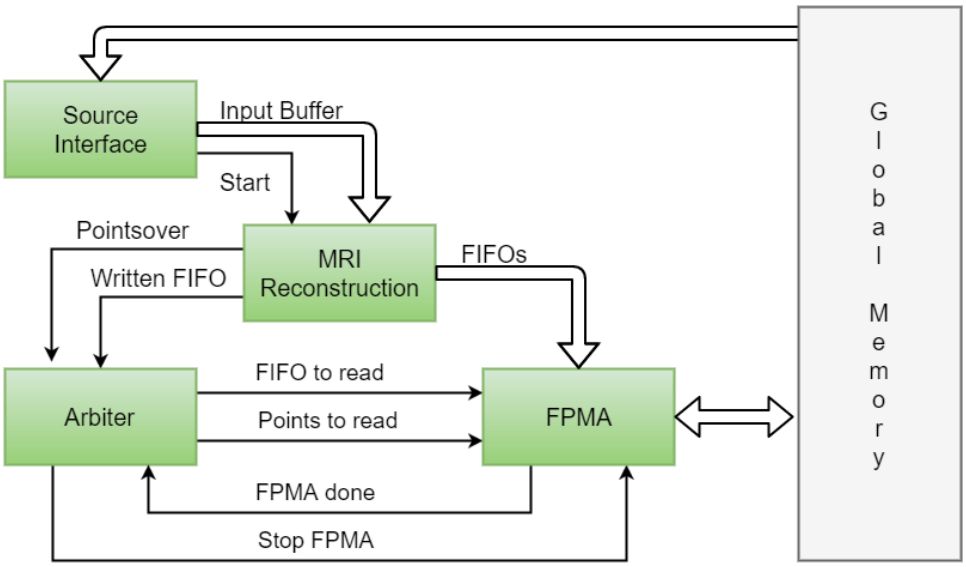


Figure 10: First OpenCL implementation, kernels block diagram

- **Source Interface**: this kernel is in charge of taking data from the host code, putting them in an internal buffer (Altera OpenCL channel extension is used for this) and to start the regridding process as soon as the first data is in the buffer.

- **MRI recon**: in order to understand this kernel the concept of tiling must be introduced. The source points space is divided into small areas called tiles, with the final purpose of reducing the number of global memory accesses. This kernel is in charge of identifying the tile a source point belong to. Since a source point could belong to more than a single tile (if it is in the border it could affect target points belonging to more tiles) it is written in all the FIFOs of the tiles it belongs to. This kernel also keeps track of the number of

source points processed and alerts the arbiter when all of them have been computed. It, furthermore, sends, for each source point, the tile (or tiles) it is identified with.

- **Arbiter**: this kernel is the "brain" of the design. It keeps track of the number of points that needs to be processed for each tile, and it decides what tile needs to be processed next (depending on a specified trajectory pattern, or sending the tile with the most points in the buffer).
- **FPMA**: stands for "Floating Point Multiplier Accumulator". It is the "worker" of the architecture.

  It is in charge of fetching the tile data from the global memory every time the arbiter is communicating to change tiles, to execute the interpolation operation, computing the contribution for each source point in the target space, and accumulating it. Once the Arbiter sends a new tile, data is written back in the global memory, and a new tile is fetched.

Once compiled, several problems came out of this design. The most critical were:

1. Global memory accesses: for space reasons in the FPGA the compiler was unable to create buffers big enough to store the source points belonging to each tile. This led to a continuous tile switching, to several memory accesses, which incredibly slows down the process.
2. Not-fixed loops for FPMA: since the number of source points for each tile was not fixed, the FPMA kernel was not pipelined (the number of iterations was changing every iterations).

The non-pipelined behavior of this kernel was, furthermore, increasing the number of clock cycles.

3. Channel cycle: the cycle created between the FPMA kernel and the Arbiter kernel created several problems for the compiler: emulating the architecture, exploiting the emulator tool provided by Altera with the OpenCL SDK. Everything seemed to be working fine. During the hardware generation phase, Altera OpenCL compiler was wrongly seeing the channels as useless, removing them from the design. It was possible to solve this problem using some directives of OpenCL.

For all the reasons above, it was not possible to obtain satisfactory results. It was an important step towards the actual implementation since it highlighted all the problems that need to be tackled in order to exploit the functionality of the compiler and of the FPGA.

4.4 Proposed Implementation

In order to design an efficient system using OpenCL it is necessary to stick to some simple "rules" as much as possible. It is fundamental that the compiler can infer the pipeline for each kernel. This implies that if a kernel has a loop, the number of iterations in this loop must be fixed prior to the compilation.

It is also very important to reduce the accesses to the global memory as much as possible. For regridding-based non uniform Fast Fourier Transform, and for a scalable architecture, this is almost impossible due to the accumulating behavior of the algorithm, that is continuously updating values in the target domain (uniform grid in k-space).

The proposed design was written following mostly the first directive which was the one that really slowed the performance in the first implementation. Even if just one kernel is not pipelined it is going to affect the performance of the entire system.

The proposed architecture also exploits, unlike the previous one, some features of the host machine to guarantee speed the efficiency of the system.

4.4.1 Host Code parallelism and Pre-processing

As mentioned before in the OpenCL the host machine is responsible for managing the communication with the FPGA, allocating and gathering all the structures. Furthermore it can also synchronize the OpenCL kernels execution invoking them when needed. The host code have been coded to add some important functionality on the system:

1. Software Pipelining: MRI processing requires subsequent sets of data to be processed sequentially. In order to improve performance in the proposed architecture the host code is able to invoke the kernels in a pipelined manner. OpenCL events are exploited so to fulfill this purpose. A more in-dept discussion can be found in Iacobucci [2].

2. Density weights Pre-computation: as mentioned in the Density Compensation Chapter, the pre-compensation requires knowledge of the MRI acquisition trajectory a priori. Furthermore it is possible to assume that an MRI machine will have fixed acquisition parameters, so the density weights computation will not change in different acquisitions of the same MRI machine. For this reason, density compensation weights can be computed once off-line and then passed directly to the FPGA as kernel arguments.

   The method chosen to compute the density compensation weights is the Voronoi diagram.

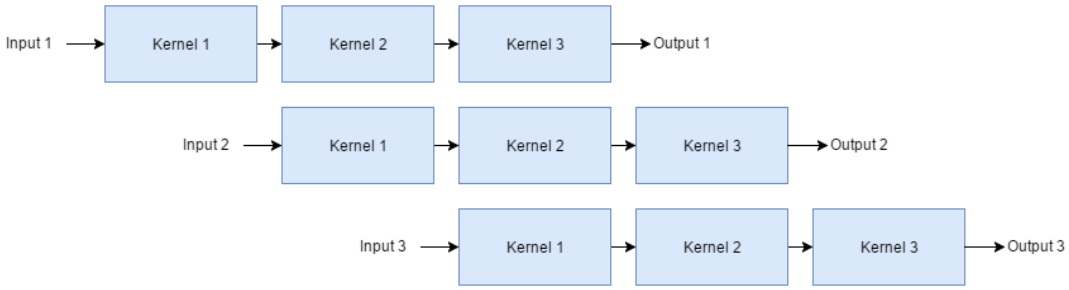


Figure 11: The Host machine is invoking the kernel following a pipelining manner

The algorithm proposed is called just once at the beginning of the pipeline and it is applied to each and every acquisition.

It directly implements the theoretical steps needed to compute the Voronoi diagram. It is possible to think of the source space as a 2 dimensional space from -0.5 to 0.5 in both the dimensions. This space is divided into a fixed number of points (the space between two different points in the source space should be lower than the minimum space between two points), and each of these points is associated with the closest source point as defined in Equation 4.

It is just an easy way to compute the Voronoi weights for each source point, but it is effective and, since there is no limitations in the execution time, it is adequate. A pseudocode of the algorithm can be found in Appendix A.

4.4.2 FPGA Accelerator

The OpenCL kernels design have been deeply influenced by the limitations given by the compiler and by the algorithm. Numerous subsequent accesses to memory are required, for

both reading and writing. To make these interactions fast enough the local memory in FPGA have to be used, although its size is limited.

The purpose of the proposed solution is to be scalable and efficient at the same time. To reach ideal scalability the amount of local memory (which is dependent on the FPGA architecture in use) should not be linked to the size of the frame processed. To avoid that the Tiling approach of the previous implementation has been extended and slightly modified to fit this approach. Tiles are no longer referred to the source points space (non uniformly sampled k-space) but to the target point domain (uniform grid k-space). The tiling division in the target domain allows no repetition of points in the queues (buffers).

Furthermore, assuming the knowledge of the sampling trajectory a priori, it is possible to estimate how many target points belong to a given tile, enabling a fixed boundary for the loop for the kernel in charge of updating the target point values. The number of tiles L can be chosen arbitrarily depending on the FPGA architecture in use. In the proposed architecture a number of tiles up to 4 have been tested. The tiling division is performed as in Figure 12.
The symmetric structure has been chosen for 2 main reasons:

1. To simplify the matrix addressing operation, during the reading and writing back operation, an addressing operation between the overall target k-space and the addressing space of the considered tile must be performed. This structure allows an easy and fast addressing operation to be made. Furthermore, the computation to see if a target point belongs to a tile or not is simple and can be easily implemented.

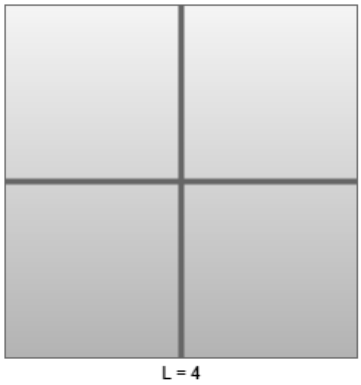


Figure 12: Tile division for L=4

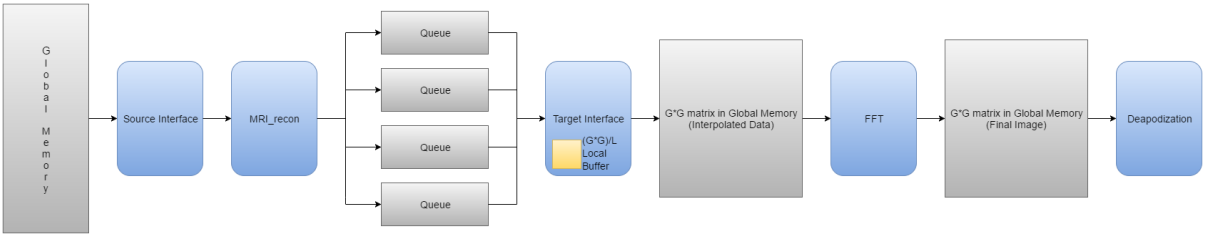


Figure 13: Block diagram for the kernels architecture for L=4. The grey blocks are data buffers in the external memory

2. For spiral and radial trajectories it is possible to roughly estimate the number of target points. Since they cover the k-space symmetrically, the number of target points for each tile can be estimated as follows:

$$TP_{tile} = \#SP \cdot (2\sigma)^2/4 \tag{4.1}$$

Figure 13 is the block diagram for the kernel architecture.

- **Source Interface**: it receives the array of source points as a parameter. It fetches one source point at a time and passes it through a channel to the Interpolation kernel. The loop is statically bounded assuming the knowledge a priori of the acquisition trajectory and of the number of source points acquired.

  The pre-density compensation can be performed also in this kernel, giving to the Source Interface kernel also the Voronoi precomputed weights array as a parameter. In that case, it will perform the division and than it will pass the already compensated data to the Interpolation kernel.

  Figure 14 highlights on the data managed by this kernel. As defined in the host code, the following data structure is defined for the Source Points array:

  - Real part: floating point value associated with the real part of the source point.
  - Imaginary part: floating point value associated with the imaginary part of the source point.
  - X coordinate: floating point value associated with the X coordinate in the k space. Originally, it varied between -0.5 and 0.5. In the host code this value is moved between 0 an 1.
  - Y coordinate: floating point value associated with the X coordinate in the k space. Originally, it varied between -0.5 and 0.5. In the host code this value is moved between 0 an 1.

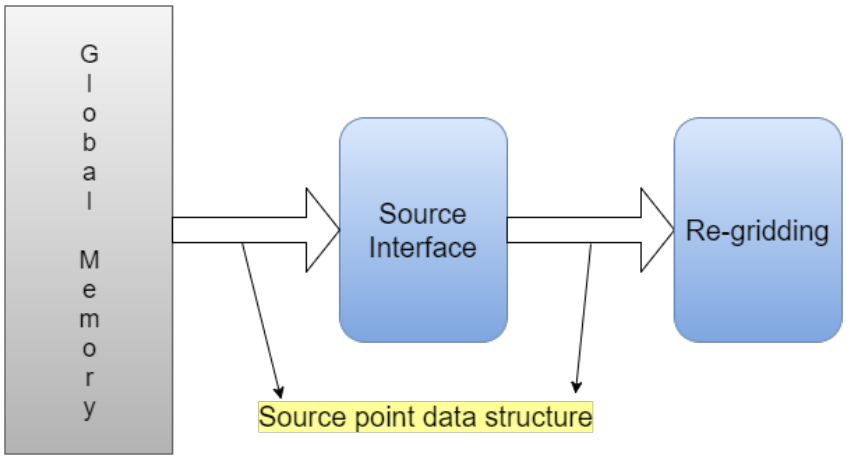


Figure 14: Source Interface step, with emphasis on the data structure managed by this kernel

- **Interpolation**: it receives source points one at a time through the channel. For each source point it generates the set of $(2\sigma)^2$ contributions to the target k-space, the index of the contribution in the target k-space. Finally, it determines the tile i into which the contribution falls and it writes the generated structure (Real part contribution, Imaginary part contribution and the target space index) into the queue belonging to that tile i.

  Figure 15 puts focus on the data managed by the Interpolation (or regrididng) kernel. It receives in a pipelined manner source points from the Source Interface Kernel. It computes the contributions and stores them in the proper queue in a data structure. The target point data structure embeds the following fields:

  – Real part: floating point value storing the real part of the contribution.

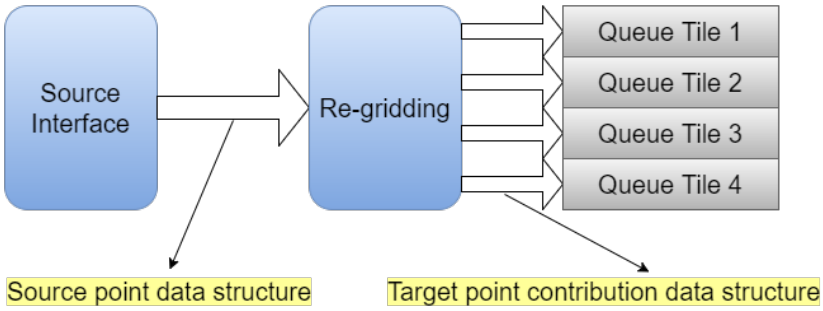


Figure 15: Interpolation step, with emphasis on the data structure managed by this kernel

- Imaginary part: floating point value storing the imaginary part of the contribution.
- Index: integer value representing the index in the target space domain of the contribution. The index is computed with respect to the grid of the target space in the following way:

$$Index = X_{TP} \cdot G + Y_{TP} \tag{4.2}$$

Where G is the number of rows and columns in the final reconstructed image, X_{TP} and Y_{TP} are the coordinates of the target point affected by the contribution.

- **Target Interface**: it waits for the Interpolation Kernel to be completed before starting. It stores a matrix of tile size in local memory initialized to all zeros. For each queue, it receives the contribution to the target k-space one at a time and sums them to the appropriate local matrix locations (addressed with respect to the global index passed in the data structure from the Interpolation kernel). In case of a post-compensation it is possible to divide by the weight in this phase so as to correctly update the target space

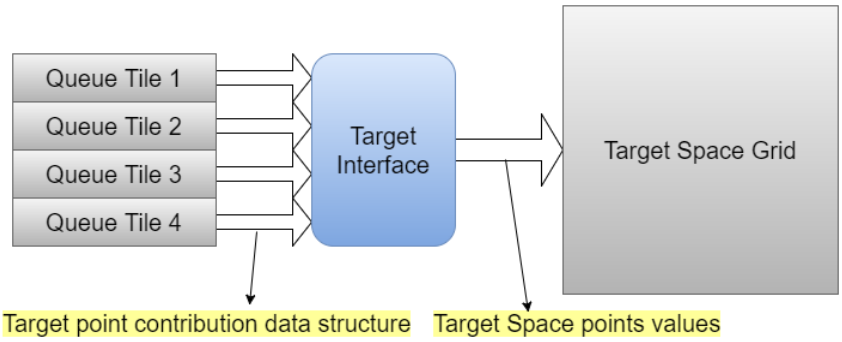


Figure 16: Target Interface step, with emphasis on the data structure managed by this kernel

every time.

The assumption of the knowledge of the number of target space contributions for each tile is made in order to define a fixed boundary for each queue a priori. At the and of each queue, the tile is written back into the memory and the local matrix is initialized to all zeros again ready for a successive tile. When all the tiles have been processed the Target k-space is finally ready and it is possible to apply the Fast Fourier Transform.

Figure 16 shows the processing of the contributions in the queues. The portion of the k space linked to a specific tile is written back, tile by tile. The data structure handling the target space points have the following fields:

- Real part: floating point value storing the summation of all the contribution for a target point.
- Imaginary part: floating point value storing the summation of all the contribution for a target point.

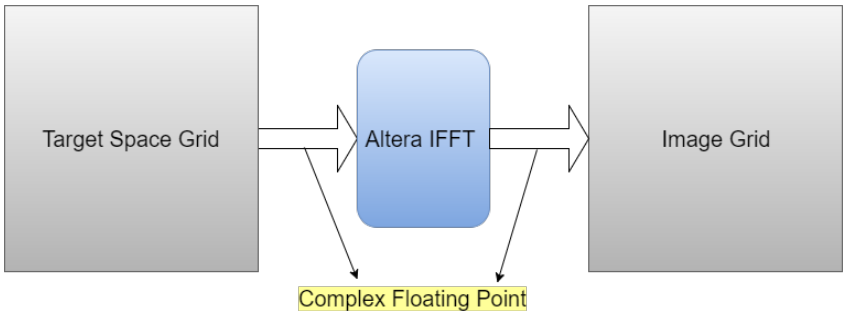


Figure 17: IFFT step, with emphasis on the data structure managed by this kernel

- **Altera IFFT Kernels**: the design can be found on Altera's website [23]. It consists of 3 kernels that perform a 1 dimensional Fast Fourier Transform. As previously mentioned, a 2 dimensional transformation can be broken down into 2 one dimensional Fast Fourier Transform applied on each row and on each column of the space. The host machine invokes the 3 kernels two times feeding them row by row in the first iteration, and column by column in the second one.

  The final result is a matrix of the same size as the target space representing the reconstructed image.

  Figure 17 shows the data managed by the Altera IFFT kernels. From a top view it takes a complex matrix of size GxG and returns a complex matrix of the same size after the transformation.

- **Deapodization**: this step can be implemented both on the fly or statically, depending on the available resources in the FPGA. In the on-the-fly implementation, coefficients

are computed in hardware applying the formula of the analytic expression for the inverse Fourier Transform of the interpolation kernel. The on-the-fly computation needs DSPs availability on the FPGA. The static implementation stores the coefficients, computed off-line, in the local memory in a vector of size G.

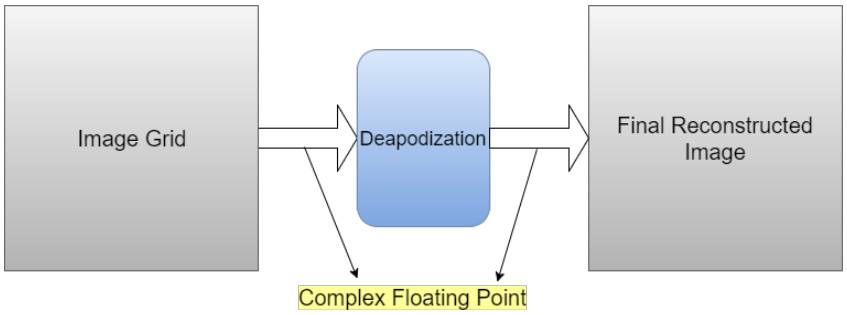


Figure 18: Deapodization step, with emphasis on the data structure managed by this kernel

Figure 18 shows the data managed by the Deapodization kernel. As in the previous case this is just a manipulation between two complex matrices of size GxG.

A flow of a source point in the architecture is described in Figure 19. The picture includes just the regridding step, which is the most complex one. A source points and its Voronoi density estimation are passed to the Source Interface Kernel by the host machine. This kernel is in charge of applying the density compensation and to pass, through a buffer, the modified

source point to the Regridding Kernel. The regridding computes the contributions to the target point uniform grid, and distributes them in the queue of the tile they belong to.

The Target Interface Kernel basically consists of several accumulators. Each accumulator represents a specific coordinate in the target space grid. The contributions generated by source points are accumulated in the specific location they belong to. This operation is repeated for each and every source point, finally, the target space will have all the contribution, and the IFFT can be applied on it.

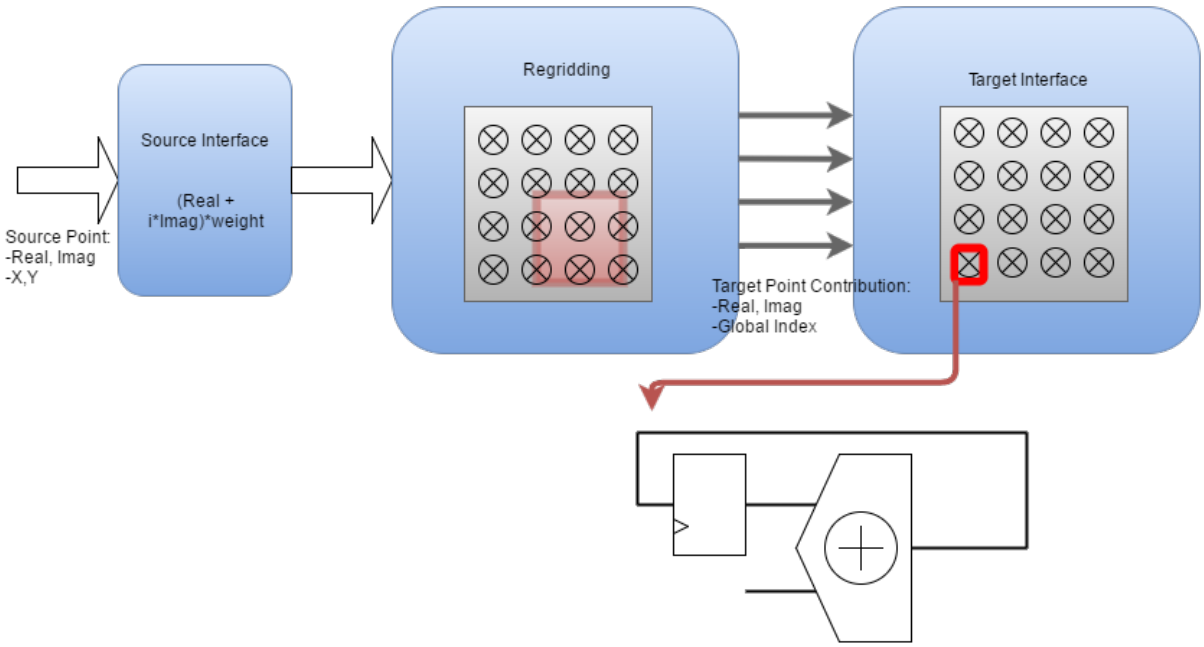


Figure 19: Flow of a source point in the regridding architecture

4.4.3 Software Reordering

The OpenCL compiler is able to translate the above kernels in a highly pipelined hardware. The only issue is for the Target Interface Kernel where the compiler is able to detect a dependency in the local memory. The same memory location could be accessed in successive iterations. To ensure data consistency the compiler has to slow down the pipeline, processing a new Target Point every 6 clock cycles. To solve this issue it is necessary to provide data in a specific order so as to avoid data dependency and to exploit a specific directive to the Altera OpenCL compiler, called *ipdev*, to make it avoid the possibility of a data dependency and to obtain a highly pipelined hardware also for the Target Interface Kernel.

Different approaches have been tried, from among the purely hardware ones, with a reordering of the points exploiting a circular buffer or a cache. Using hardware approaches, it was not possible to reduce the number of the pipeline clock cycles without loss in accuracy of the reconstructed imaged. Assuming that the sampling trajectory is known a priori, it is possible to apply a software reordering once off-line and then apply the read pattern to all the successive frames. To avoid data dependency points must be processed by the Interpolation kernel in such a way that, for each queue of contribution to the tiles, contribution to the same target point are allowed once every X contributions. In this case, since a new data is fed into the pipeline every 6 clock cycles, X should be at least 6.

The off-line algorithms pre-determines the order in which data should be read to the file to avoid dependency. It follows a greedy approach so as to have a reasonable execution time. For each source point, target points expected to be affected are computed. If all the contributions

are not already in the latest contributions for each queue, the source point is inserted and all the contributions are put in the latest contribution buffers. If there are no points satisfying this condition, a filler point is inserted.

A filler point is a source point placed at the edge of the image for each tile. In order to reduce the number of filler points inserted, source points requiring the smallest number of them are inserted in the queue.

Although this algorithm increases the number of source points the total number of filler points inserted is so small that it is negligible compared with the total number of source points. Furthermore, the gain in terms of performance overcomes the time penalty introduced. A pseudocode and a more detailed explanation of the technique can be found in Iacobucci et al. [2].

4.5 Non-Scalable Implementation

The above system is scalable and can be implemented in different FPGAs without looking at the available space. It can, in fact, be adapted to fit the available space introducing a further tile division. At the same time, if the available space is enough to fit the target space, the tile division can be avoided reducing the complexity of the code and obtaining even better performance.

The Non-scalable approach is a simplification of the scalable one when there is enough local memory in the FPGA to fit all the points of the target space. The no-tiling approach simplifies a lot the architecture since several steps of the algorithm are not needed anymore:

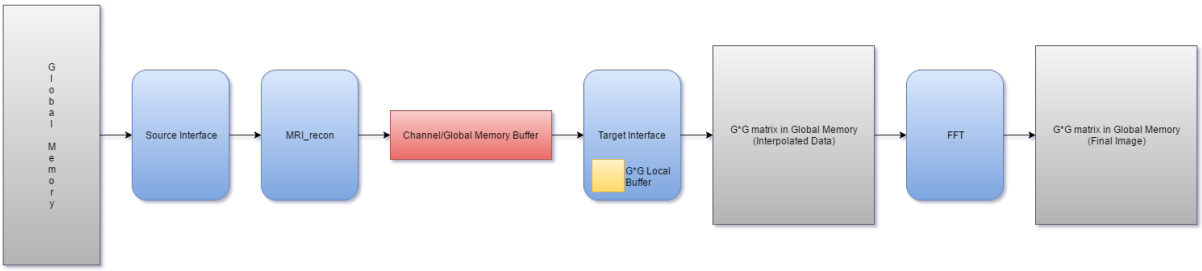


Figure 20: Block diagram for the non scalable implementation. Red block can be implementend through Altera OpenCL channel.

- The regridding step is simplified. It is not needed anymore to identify the queue a specific target point update belongs to. Thus, the logic resources used are much less.
- Since it is no need to switch tiles anymore, the Target Interface Kernel can be executed in parallel to the regridding step without waiting for the end. This significantly improves the performance. The communication between the Regridding Kernel and the Target Interface one can be implemented using the Altera OpenCL channel extension, or through global memory.

This implementation is strictly dependent on some important factors of the image reconstruction. The use of local memory is dependent on the target image size. The target image size depends on the image acquisition of the MRI machine and on the chosen oversampling factor.

4.6 Fixed Point Implementation

One of the most interesting features of the Altera OpenCL compiler is the native support for floating point operations. This is one of the most important advantages with respect to any VHDL/Verilog designed architecture. It is, in fact, tedious to implement floating point operation with Hardware Description Languages even using Intellectual Properties and design provided by the vendor (Altera includes in Quartus software several intellectual properties to help FPGA designer).

However, when programming an FPGA, fixed point operations lead to a simple hardware and to better performance. Appendix B briefly describes how floating point and fixed point arithmetic work, highlighting advantages and disadvantages of both in terms of precision, performance and hardware resources utilized.

The OpenCL standard does not support fixed point data type. This means the fixed point data types must be implemented using integer data types. The advantage of using VHDL/Verilog in describing fixed point data types is the liberty that the designer has to select exactly the number of bits needed to correctly represent a value. In OpenCL there is no such liberty, since the supported data type are 8-16-32 or 64 bits resolution. Appropriate masking operation can, however, be implemented so that the tool can perform optimizations to conserve hardware resources.

The fixed point architecture is pretty similar to the proposed one. The only difference is in the adopted arithmetic, which will exploit only fixed point numbers leading to a much simpler hardware implementation.

4.6.1 Source Point Data Structure

As we already stated in the Proposed Implementation Section, the Source Point Data structure is composed of the following elements:

- X coordinate of the source point in the k-space. This is expressed as a floating point varying from -0.5 and 0.5.
- Y coordinate of the source point in the k-space. As the above one it is expressed as a floating point varying from -0.5 to 0.5.
- Real part of the source point. Expressed as a floating point.
- Imaginary part of the source point. Expressed as a floating point.

These floating point variables must be properly adapted to integer values, and the kernel that manages these data must me properly modified. The data is modified from floating to fixed point directly in the Host code, where also the fixed factor is impressed. The Source Point Data coordinates are handled differently with respect to the real and imaginary values. X, and Y coordinates are normalized in the range 0 to 1 in the host code. The regridding kernel adapts the scale from 0 to 1 to 0 to G. This allows one to define a bound on the number of bits of the integer part of the fixed data representation:

$$IntegerBits = \log_2(G) + 1 \tag{4.3}$$

So for a target picture of 256*256 9 bits are necessary, for a target picture of 512*512 10 bits, while for a 1024*1024 11 bits are necessary. To have a acceptable precision in the decimal representation 32 bit integer number are used.

In the worst case scenario, for an impractical target frame size of 1024*1024 the precision of the decimal part will be:

$$\frac{1}{2^{21}} = 4.76E^{-7} \tag{4.4}$$

Which is more than enough considering the further approximation to the nearest neighborhood performed in the regridding kernel.

On the other hand, real and imaginary values are not bounded by maximum and minimum values, making the choice of the fixed factor much more difficult, and deeply linked with the dataset used. Furthermore, they have to be multiplied by the interpolation kernel value, generating a contribution for a target space grid point. More contributions will then be accumulated generating finally the complete target space grid.

While the multiplication is never for a number greater than 1, one cannot determine a reasonable maximum value in the accumulation to bound the integer part. The factors proposed in the following worked well for the datasets tried, but the same cannot be guaranteed for other datasets. These have to be properly tuned for each MRI machine, possibly knowing the maximum real and imaginary value that can be sampled.

To keep a comparable precision with respect to the floating point architecture a 32 bit integer value has been used. The decimal bits are set to 20, while the integer bits to 12. This

should guarantee enough precision for the decimal part, and enough range to avoid overflow. Specifically:

$$\frac{1}{2^{20}} = 9.53E^{-7} \approx 1E^{-6} \tag{4.5}$$

$$-2048 < integer < 2047 \tag{4.6}$$

4.6.2 Kernels Modifications

Kernels have been modified to handle fixed variables. The most significant changes have been performed in the Regridding, FFT and Deapodization kernels.

1. **Regridding**: the kernel vector used to compute the kernel coefficient for the interpolation is properly adapted from floating point to fixed. A factor of 7 has found to be enough to guarantee good results in terms of accuracy. The management of the coordinate values are properly adapted, and a fixed point round function has been created so to handle the nearest-neighborhood computation in the interpolation phase.

2. **FFT**: Altera IFFT kernels are adapted so as to receive and produce a fixed point matrix. Some modifications to the kernel have been performed to avoid floating point operations. In the algorithm used for the Fast Fourier Transform computation, a look-up table with sin and cosin floating points is stored. These values have to be properly adapted into integer variables.

   Also in this case the factor found to guarantee good accuracy results is 7.

3. **Deapodization**: Also the deapodization kernel needs to be modified to properly handle a fixed point architecture. In that case a lookup table with the deapodization vector

(computed from the used interpolation function) has been converted from floating point to fixed point.

The factor found to be the most effective is 10.

4.6.3 Overflow Prevention Techniques

Extra care must be taken when using fixed point arithmetic, mostly for multiplications and divisions.

1. **Kernel Coefficient Multiplication**: it a multiplication between a number with 20 decimal bit (real and imaginary values) and the interpolation kernel coefficient with 14 decimal bit. The result is temporarily stored in a 64 bit variable and then is shifted back by 14 bit into a 32 bit variable.
2. **FFT Multiplication**: as in the previous case, the result is stored in a temporary 64 bit variable and than it is shifted back by 7 bit.
3. **Deapodization Coefficient Division**: after the Fast Fourier Transform the values are divided by the deapodization coefficients. No further operation is required.

CHAPTER 5

RESULTS

This chapter will focus on the result obtained with the proposed architecture from the point of view of both the accuracy of the reconstructed image and the performance of the accelerator. The analysis will start with the accuracy results obtained with different approaches of density compensation (pre-computed post compensation, on-the-fly post compensation, full Voronoi pre-compensation, reduced Voronoi pre compensation) and then on the performance results for the different approaches of above that can simplified into 4 groups (computed on the fly compensation, compensation of the source points in source interface, compensation of the target point in target interface, compensation applied in the source code).

Then the results of the proposed architecture in term of performance and accuracy are shown, along with the improvement obtained for the non-scalable solution for the 128 and 256 pixel input source.

Finally the results of the fixed point architecture are compared with the proposed one in term of performance, accuracy and resource utilization.

5.1 Testing System

A Lenovo ThinkStation P500 with 32 GB of DDR4 RAM, 8 Intel Xeon E5-1629 CPUs running at 3.50GHz and with CentOs 6.8 as operating system has been used for testing. The target FPGA was an Altera Arria 10 GX, which exchanges data with the host machine through PCIe16 connection. The OpenCL code was compiled using the Altera OpenCL Compiler Version 15.1.

5.2 Architecture Parameters

The tested implementation uses an oversampling factor $\alpha = 2$, a Kaiser-Bessel windowed function as the interpolation kernel, with a convolution windows size $\sigma = 2$. The parameter of the Kaiser Bessel was chosen according to the formulas from [24]:

$$C(k_x) = \frac{G}{2\sigma} \cdot I_0 \left(\beta \sqrt{1 - \left(\frac{2Gk_x}{2\sigma} \right)^2} \right) \quad for |k_x| \leq \frac{2\sigma}{G} \tag{5.1}$$

$$\beta = \pi \sqrt{\frac{(2\sigma)^2}{\alpha^2} \left(\alpha - \frac{1}{2} \right)^2 - 0.8} \tag{5.2}$$

Equations 5.1 and 5.2 show the analytic expression for the kernel and for the parameters computation.

The Kaiser-Bessel function has been uniformly sampled in $S \cdot W$ points, and the kernel value is chosen with the nearest neighbor interpolation technique, so that the actual equation for the used kernel is:

$$\hat{C}(k_x) = C(k_x) * Gate(SGk_x) \tag{5.3}$$

Given this choice for the convolution kernel, its IFFT is:

$$c(x) = \frac{\sin\sqrt{\frac{\pi 2\sigma x}{G^2} - \beta^2}}{\frac{\pi 2\sigma x}{G^2} - \beta^2} \cdot \frac{\sin(SGk_x)}{SGk_x} \tag{5.4}$$

and it has been used for on-the-fly computation of Deapodization coefficients. S is a coefficient calculated to impose a maximum nearest-neighbor interpolation error of 10^{-2}, according to the formula described in [24]:

$$max(nearest\_neighbor\epsilon) = \frac{0.91}{\alpha S} \tag{5.5}$$

The equations above have been generalized to work for the 2-dimensional case.

5.3 Accuracy Metrics

The accuracy metrics are used to evaluate the image reconstruction are 2: PSNR and SSIM.

Peak signal-to-noise ratio (PSNR) evaluates the reconstruction of a signal by dividing the maximum power possible in the signal with the power of the noise that is corrupting that signal [25]. It is expressed in terms of a logarithmic scale.

$$PSNR = 20log_{10}\left(\frac{MAX_I^2}{MSE}\right) \tag{5.6}$$

where MSE is the mean squared error defined as:

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2 \tag{5.7}$$

An higher PSNR generally indicates that the reconstruction has a higher quality. Usually typical values for good accuracy in the image is between 30 and 50dB.

The Structural Similarity Index (SSIM) unlike to the more classical PSNR takes into account how a human being perceives an image [26]. It focuses on the structure of the image, that is the most critical parameter in MRI, since it requires one to reconstruct a picture in such a way that doctors can see what they need to diagnose a patient [27]. It is defined as follows:

$$SSIM(P) = \frac{2\mu_1(P)\mu_2(P) + C_1}{\mu_1(P)^2 + \mu_2(P)^2 + C_1} \cdot \frac{2 \cdot cov(P) + C_2}{s_1(P)^2 + s_2(P)^2 + C_2} \tag{5.8}$$

where:

- μ_1, μ_2 are the mean values of the two images over a small window around the point P.
- s_1, s_2 are standard deviations of the two images over the same window.
- cov is the covariance over the same window.
- C_1, C_2 are regularization constants.

This index measures the structural similarity between two images, and it varies between -1 and 1. Two nearly identical images will result in an index close to 1.

5.4 Performance Metrics

Performance are evaluated following some defined rules:

- The number of source points is the size of the input image. For a 128x128 input image, 128x128 source points will be passed to the architecture.

- The oversampling factor is set to 2. So the target frame will have double the size of the input frame.

The actual metric used to evaluate performance is frames per second (fps), the number of frames that the architecture is able to process in a second. This is not calculated computing how many times a frame can be processed in a second, but providing the same input to the architecture several times so to verify also the effectiveness of the software pipelining.

5.5 Resource Utilization Metrics

The resource utilization is expressed as a percentage of the utilized resource in the Altera Arria 10 GX.

Specifically the total available hardware in the FPGA is:

TABLE I: TOTAL RESOURCE AVAILABLE IN ARRIA 10

LE	855289
FF	1081705
RAM	2714
DSP	1518

The "Util" section includes an overall percentage of the used resources in the FPGA.

5.6 Density Compensation Results

As stated in the introduction to this chapter, different compensations have been tested in order to obtain the best results. In the density compensation chapter, two ways of compute the

density compensation have been analyzed: post-compensation (applied directly on the uniformly sampled target space), and pre-compensation (applied on the source non-uniformly sampled source points).

We already stated that the post-compensation leads to some problems with aliasing and requires a specific sampling step in the target space. It is also the only one that does not require the knowledge a priori of the sampling trajectory and that can be computed on the fly. Pre-compensation is the most used typology of compensation in the MRI field, but it requires the knowledge of the sampling trajectory and, sometimes, complex computations.

Accuracy results

In order to measure the accuracy results of our architecture a dataset provided by Stanford University have been used. This dataset is well-suited to test the correct working of an MRI reconstruction algorithm and architecture, due to precise and sudden borders within the final picture.

The picture is recurrent in the literature of MRI reconstruction and it is also appropriate to understand how the choice of the density compensation methodology affects the accuracy of the results.

In order to focus only on the density compensation, the algorithm does not the deapodization step.

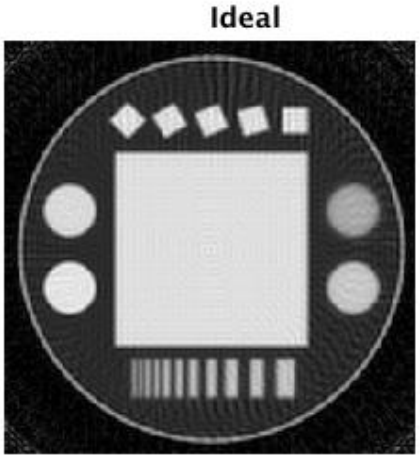


Figure 21: Ideal Reconstructed Image for Stanford Dataset [28]

First of all, the importance of the density compensation step can be highlighted by the image reconstructed without that step, as it is possible to see in Figure 22. The reconstructed image appears blurry and it is almost impossible to see clearly what it is showing. The fact that the image is recalling somehow the ideal reconstructed image indicates that the chosen kernel interpolation function is properly working, along with the width of the interpolation window and the sampling step of the uniform target space.

Depending on the type of density compensation chosen there are different algorithms that can be applied exploiting some assumptions. For the post compensation:

Figure 22: Image reconstructed without the density compensation step. Data from [28].

1. **Post-compensation on-the-fly**: this is exploiting the fact the the post compensation does not require a knowledge a priori of the sampling trajectory.

2. **Precomputed post-compensation**: it is safe to assume that the trajectory is known a priori (thus, completely removing the only advantage of the post-compensation with respect to the pre-compensation). With this assumption is possible to compute the post compensation once before the execution of the accelerator. The oversampling factor adopted in the FPGA architecture should be already taken into account when computing the coefficients. The post-compensation refers to the target domain space, generating a weight for each target point. The number of target points depends on the size of the source frame

and on the oversampling factor chosen.

The division for the weight is performed for each and every contribution and thus, causes a loss in the accuracy of the reconstructed image.

For the pre-compensation, we tried to tackle the most general solution possible. So Voronoi methodology has been used.

1. **Full Voronoi**: in this case the Voronoi diagram is computed for all the source points in the space. In order to use the pre-compensation typology the a priori knowledge pf the sampling trajectory is already assumed.
2. **Half Voronoi**: in this case the Voronoi diagram is computed just for the source points close to the center of the k-space. The area of the polygon associated with each source point, for radial and spiral trajectories, progressively increases with the distance to the center. The Voronoi diagram is computed until the area of the polygons is within a certain maximum value, then all the other polygons are assigned to this value.

   The value chosen in order to stop the Voronoi computation within a limited number of points, obtaining acceptable accuracy results is 0.1.

Figure 22 shows the image reconstructed using the different approaches described above. For a human eye it is almost impossible to notice any difference between the two reconstructed images, even if it can be noticed that the image reconstructed using the Voronoi approach is a

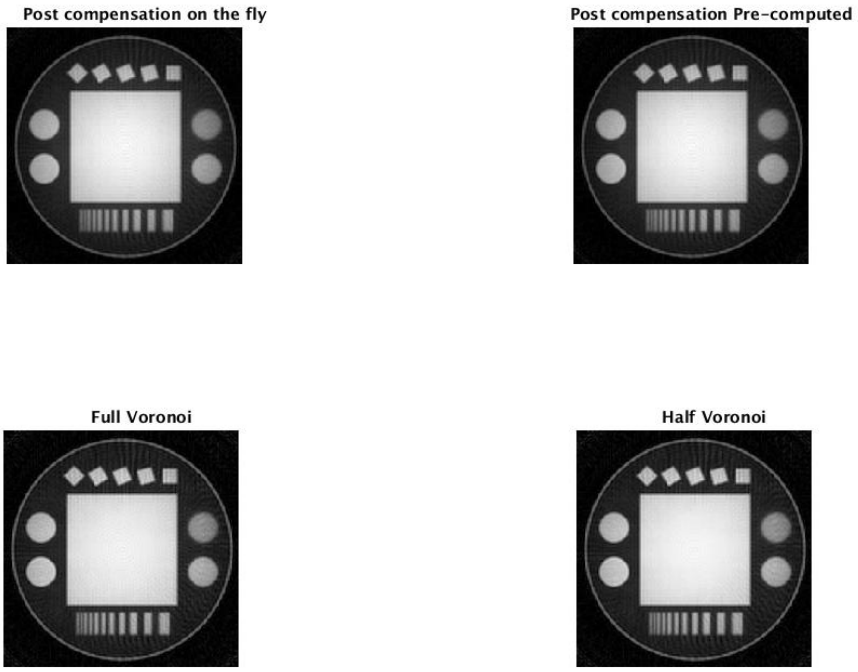


Figure 23: Stanford image reconstructed using different typologies of density compensation. Data from [28].

TABLE II: DIFFERENT SSIM AND PSNR FOR THE DENSITY COMPENSATION METHODOLOGIES ADOPTED WITH RESPECT TO THE IDEAL RECONSTRUCTION FOR STANFORD DATA SET.

	SSIM	PSNR
No Compensation	0.2664	14.0681
Post Compensation precomputed	0.7579	21.0569
Post Compensation on-the-fly	0.7584	21.0682
Full Voronoi	0.8318	23.1444
Half Voronoi	0.8168	22.5762

bit clearer than the one reconstructed adopting the Post-compensation.

Results obtained in Table II are quite expected. The quality of the reconstruction without density compensation is low. Post-compensation results are comparable, the minor difference is given by the loss of accuracy in the sum of divisions. Pre-compensation is, as expected, the one that gives the best results. The Half Voronoi methodology proves its effectiveness, highlighting that the computation of the areas of a limited number of polygons in the Voronoi diagram and the assigning the rest to the ultimate value does not significantly affects the reconstruction.

Performance Results

As we said in the introduction to the chapter, performance and resource utilization will also be affected by the choice of the density compensation methodology. For sake of simplicity all the performance results are obtained with a number of source points equal to the pixels in the target image, with a convolution window of size 2, and with an oversampling factor 2.

TABLE III: DIFFERENT SSIM AND PSNR FOR THE DENSITY COMPENSATION METHODOLOGIES ADOPTED WITH RESPECT TO THE IDEAL RECONSTRUCTION FOR A RADIAL TRAJECTORY DATA SET.

	SSIM	PSNR
No Compensation	0.2283	8.0173
Post Compensation precomputed	0.775	18.4369
Post Compensation on-the-fly	0.782	18.4692
Full Voronoi	0.9128	21.243
Half Voronoi	0.8272	19.5064

1. **Post-compensation on the fly**: this option, the only one that does not assume the knowledge a priori of the sampling trajectory, heavily increases the number of utilized resources (requiring a further buffer in the target interface).

2. **Pre-compensation in Source Interface**: this solution introduces a further step in the Source Interface Kernel. The well pipelined hardware generated by the compiler, assures good results from a performance point of view.

3. **Post-compensation precomputed**: this option requires a division by the coefficients in the Target Interface Kernel. This division is well pipelined and does not heavily affect the performance.

4. **Pre-compensation in the Host Code**: this option is, theoretically speaking the best. Input source points fed to the architecture are already compensated and no further operation is required in the FPGA.

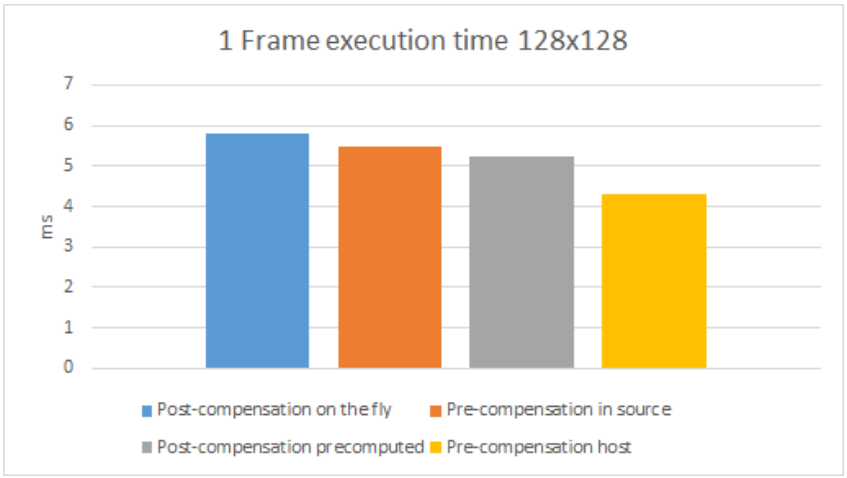
The following table shows the resource utilization and the execution time for each case described above. As expected, the last option gives the best performance with less hardware resource utilization.

TABLE IV: RESOURCE UTILIZATIONS FOR A 128X128 INPUT, OVERSAMPLING FACTOR = 2.

128x128	LE	FF	RAM	DSP	Util
Post-compensation on the fly	24%	36.94%	41.68%	62.31%	60.84%
Pre-compensation in source	21.77%	33.16%	35.27%	58.03%	54.82%
Post-compensation precomputed	22.05%	33.41%	36.30%	58.03%	55.32%
Pre-compensation host	21.64%	32.92%	34.35%	57.37%	54.45%

TABLE V: RESOURCE UTILIZATIONS FOR A 256X256 INPUT, OVERSAMPLING FACTOR = 2.

256x256	LE	FF	RAM	DSP	Util
Post-compensation on the fly	26.23%	38.45%	82.85%	62.12%	65.89%
Pre-compensation in source	23.98%	35.48%	70.22%	59.09%	56.54%
Post-compensation precomputed	24.88%	35.72%	71.36%	59.07%	57.68%
Pre-compensation host	23.72%	33.50%	70.35%	58.78%	56.32%



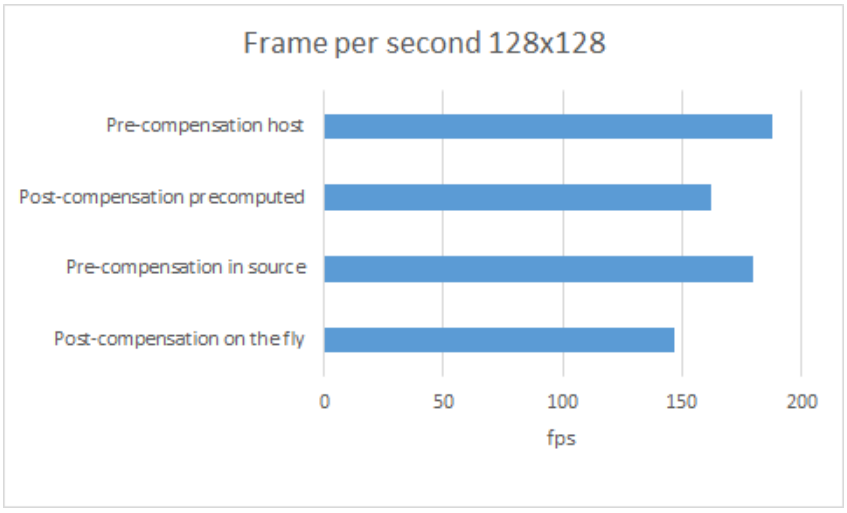


Figure 24: Performance results for a 128x128 pixel input image.These results have been collected for the research, for this reason they can be also found in the Iacobucci [2]

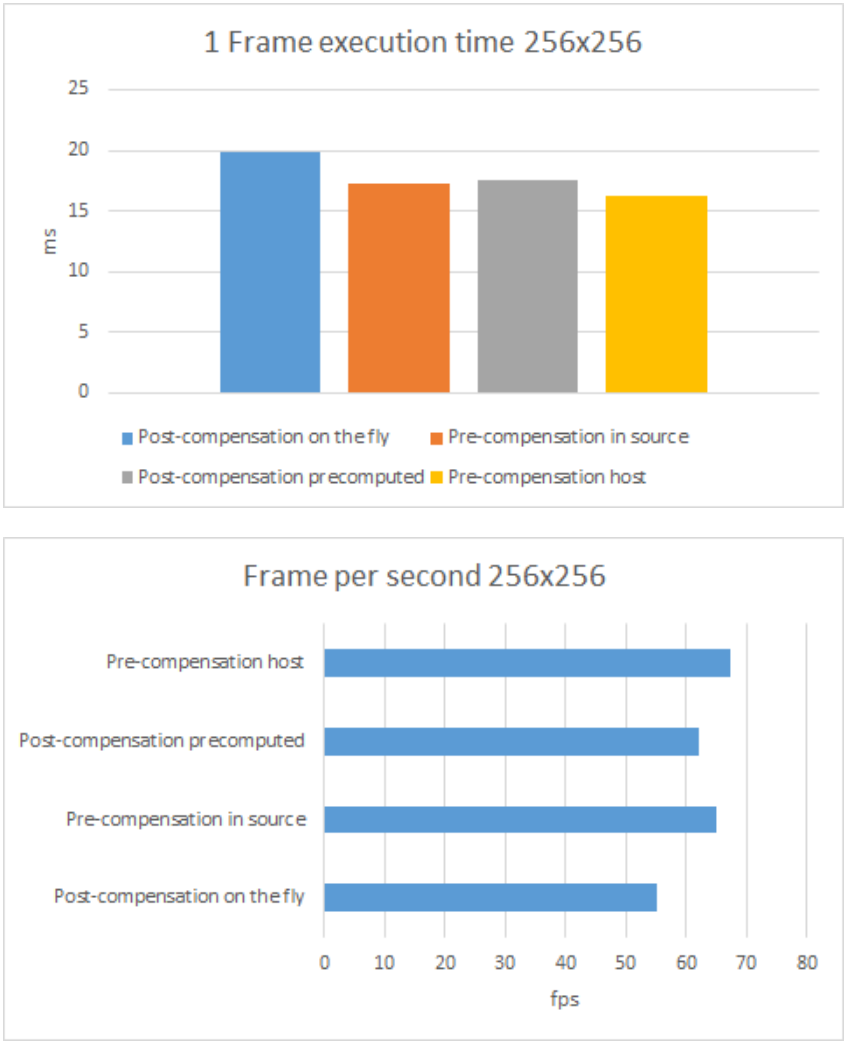


Figure 25: Performance results for a 256x256 pixel input image.

TABLE VI: RESOURCE UTILIZATIONS FOR A 512X512 INPUT, OVERSAMPLING FACTOR = 2.

512x512	LE	FF	RAM	DSP	Util
Post-compensation on the fly	37.23%	45.67%	98.45%	72.52%	85.34%
Pre-compensation in source	35.94%	42.56%	91.22%	70.34%	75.87%
Post-compensation precomputed	36.56%	41.98%	93.45%	70.45%	73.69%
Pre-compensation host	34.78%	41.12%	91.57%	68.34%	71.68%

5.7 Proposed Architecture Results

The proposed architecture embeds the best density compensation approach (Voronoi density compensation computed in the host code), and adds the deapodization step to further improve the accuracy of the image reconstruction. Accuracy results are, again measured in SSIM and PSNR. Perfromance is compared to similar architectures proposed in the literature.

It was not easy to find suitable architectures to compare our architecture with. Mostly of the work proposed in the literature focuses only on the regridding step, which is the one that requires the most computational power and hardware resources, completely ignoring the importance of the density compensation and of the deapodization.

Accuracy Results

The SSIM and PSNR obtained, using the proposed architecture on the dataset introduced above are the following:

TABLE VII: ACCURACY RESULTS FOR THE PROPOSED ARCHITECTURE. THIS IS INCLUDING ALSO THE DEAPODIZATION STEP, OTHER THAN A FULL VORONOI DENSITY COMPENSATION APPLIED IN THE HOST CODE.

	SSIM	PSNR
Stanford	0.9585	34.5302

Performance Resutls

In order to have a reasonable performance comparison with other architectures proposed in the literature, the design has been compiled in two different fashions, so to include:

- Regridding step only, to compare results with the FPGA solution proposed by Kestur et al. in [29] and with the CPU solution proposed by Sorensen et al. in [30].
- Density Compensation, Regridding, IFFT and Deapodization compared with the results obtained by Sorensen et al. in [30].

The throughput is measured in frame per second (fps), where the size of the frame is given by the frame of the input image multiplied the chosen oversampling factor.

The results show a marked improvement with respect to the regridding solutions highlighting the efficiency of the OpenCL FPGA implementation.
Results with the GPGPU solution in [30] are, on the other hand, comparable, indicating the further work that is required in order to improve the architecture.

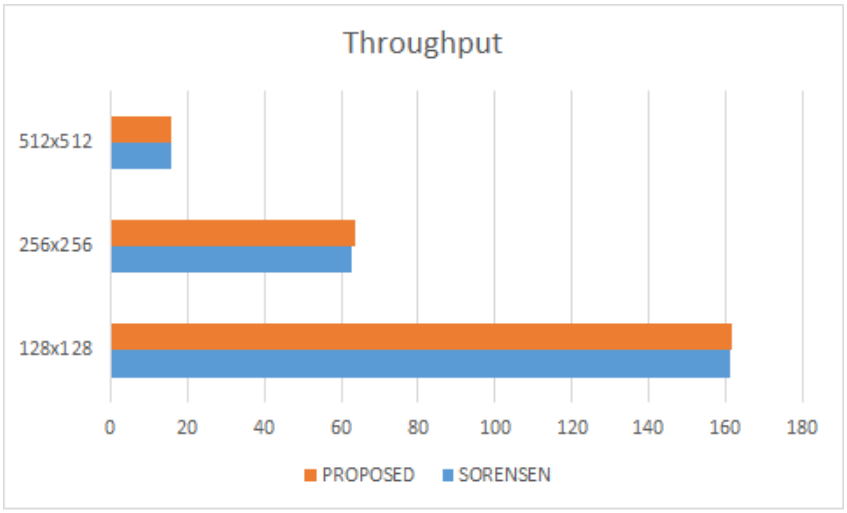


Figure 26: Throughput of the entire process (Density Compensation + Regridding + Deapodization). The results are compared with the one obtained in with a GPU [30]. These results have been collected for the research [1], for this reason they can be also found in the Iacobucci [2]

5.8 Non Scalable Implementation Results

The non scalable implementation guarantees better results thanks to the use of an Altera channel between the regridding kernel and Target Interface Kernel. This allows to start accumulating the contributions in the target space while the regridding phase is still going on.

The performance gain is described in the following tables.

TABLE VIII: PERFORMANCE IMPROVEMENT FOR THE NON SCALABLE ARCHITECTURE, 128X128 CASE.

128x128	1 Frame	fps
Scalable Architecture	4.31	187.61
Non Scalable Architecture	3.76	224.01

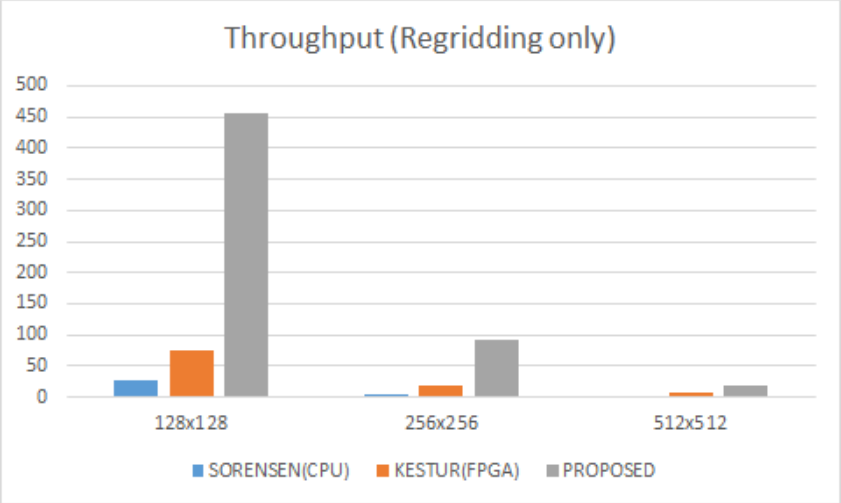


Figure 27: Throughput of only the reconstruction (Regridding) phase. Results are compared with Sorensen et all. [30] with a CPU, and Kestur at all. with an FPGA [29].These results have been collected for the research [1], for this reason they can be also found in the Iacobucci [2]

TABLE IX: PERFORMANCE IMPROVEMENT FOR THE NON SCALABLE ARCHITECTURE, 256X256 CASE.

256x256	1 Frame	fps
Scalable Architecture	16.27	67.23
Non Scalable Architecture	14.19	76.32

Just the 128x128 and 256x256 cases are illustrated, since the 512x512 case requires, for an oversampling factor set to 2, a buffer that is not going to fit in the local memory available in the FPGA.

This architecture can be easily compared with the solution proposed by Li et al. in [31]. They describe a similar implementation using Labview and obtaining good performance results but tied to the memory available on the FPGA chip considered. At the same time they do not clearly mention the size of the target image, neither the use of oversampling, and the number of source points used to reconstruct the image. The claimed result is 400 fps, computed dividing 1 second with the time to execute one frame.

The results showed in this document are the ones actually returned feeding the hardware with the same data-set over and over. In the 128x128 case the pipelining does not improve the performance, and, actually, the time to load into the FPGA a new dataset is greater than the processing time of the FPGA. For small images the bottleneck in terms of performance can be considered the Host-compute unit communication.

5.9 Fixed Point Implementation Results

As described in the Implementation chapter, fixed point arithmetic leads to a considerably improvement in term of resource utilization. The logic components to accomplish mathematical operations, such as multiplication and division, are much fewer.

The reduced hardware utilization can lead to possible new features in the design that will be discussed in the next chapter.

Accuracy Results

The results obtained with the fixed point implementation are compared with the ideal reconstruction of the image. PSNR and SSIM are used, as before, to quantify the reconstruction. Accuracy results for the Stanford data set are showed in the the table and graphically.

The Fixed point implementation is not able to keep the same accuracy in the reconstructed image. However, it is possible to tune some parameters accordingly to the input image in order to have better accuracy in the reconstruction.

TABLE X: ACCURACY COMPARISON BETWEEN THE PROPOSED FLOATING POINT IMPLEMENTATION AND THE FIXED POINT.

Stanford	SSIM	PSNR
Fixed Point	0.72	18.87
Floating Point	0.95	34.53

Figure 28: Graphic comparison between the ideal reconstruction and the one obtained with the Fixed point implementation . Data from [28].

Performance Results

Performance results are showed in Figure 29. Although comparable results were expected with respect to the floating point implementation (exactly the same operations are performed in both the two architectures), the fixed point implementation is slower.

This is due to the fact that the frequency of the hardware is decreased from 180MHz to 145MHz. To obtain comparable performance it is possible to exploit the reduced hardware resources utilized and increase the number of points processed at the same time in the Target Interface kernel. For processing 16 contributions to the target space at the time the execution time is comparable to the floating point implementation.

Resource Utilization Comparison

Resource utilization is compared with the least hardware demanding floating point implementation (density compensation in the host code). As expected, the improvement in hardware

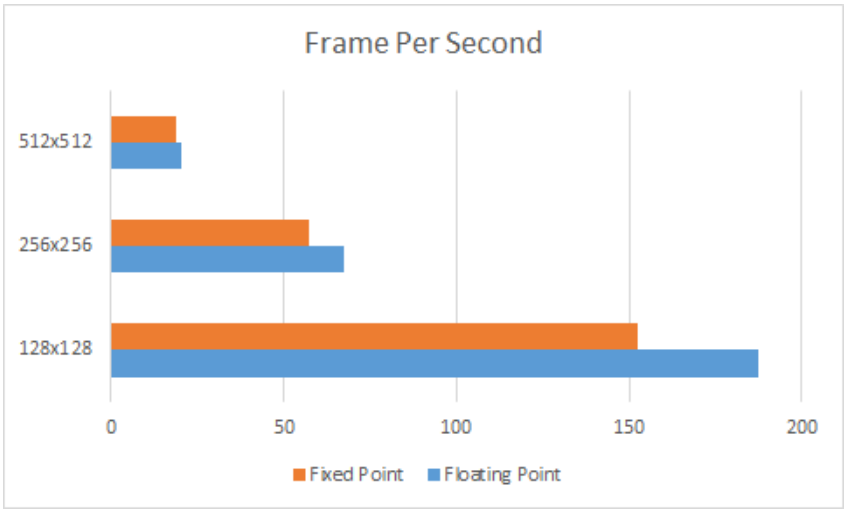


Figure 29: Frame per second comparison between Floating and Fixed point architecture

resources actually utilized is impressive. The only comparable parameter is the local memory, which is more or less the same. The implementation still requires a local buffer of the same size of the floating point implementation.

TABLE XI: COMPARISON FOR THE 128X128 CASE USING FLOATING AND FIXED POINT ARITHMETIC.

128x128	LE	FF	RAM	DSP	Util
Fixed Point	6.25%	9.91%	33.06%	16.46%	16.17%
Floating Point	21.64%	32.92%	34.35%	57.37%	54.45%

TABLE XII: COMPARISON FOR THE 256X256 CASE USING FLOATING AND FIXED POINT ARITHMETIC.

256x256	LE	FF	RAM	DSP	Util
Fixed Point	6.38%	10.07%	70.80%	19.63%	16.46%
Floating Point	23.72%	33.50%	70.35%	58.78%	56.32%

TABLE XIII: COMPARISON FOR THE 512X512 CASE USING FLOATING AND FIXED POINT ARITHMETIC.

512x512	LE	FF	RAM	DSP	Util
Fixed Point	7.12%	15.34%	92.67%	24.45%	18.62%
Floating Point	34.78%	41.12%	91.57%	68.34%	71.68%

CHAPTER 6

FURTHER APPROACHES

The proposed architecture is able to process MRI data in an efficient way, performing an image reconstruction with high accuracy. An improvement in term of performance is possible.

6.1 Modified FFT Kernels

The synchronization between kernels performed in the host code slows down the process, leaving the FPGA waiting for new data to be processed most of time. This problem is highlighted between the regridding kernel and the Target Interface one, between the ladder and the FFT, and between the FFT and the deapodization. It is possible to solve this issue using more hardware resources making the accelerator obtain much better results in term of performance.

The synchronization between kernels can be managed using Altera OpenCL channels, avoiding to wait for the host. FFT kernels exploit NDrange kernel, multiplicating the execution during the fetch and writing back execution. In order to avoid a loop in the same kernel, FFT must be duplicated and the synchronization can be performed using channels.

The execution time for a frame, for an input image of 128x128 is reduced by 1.4 ms leading to a huge improvement in performance, as it is possible to see in Figure 30.

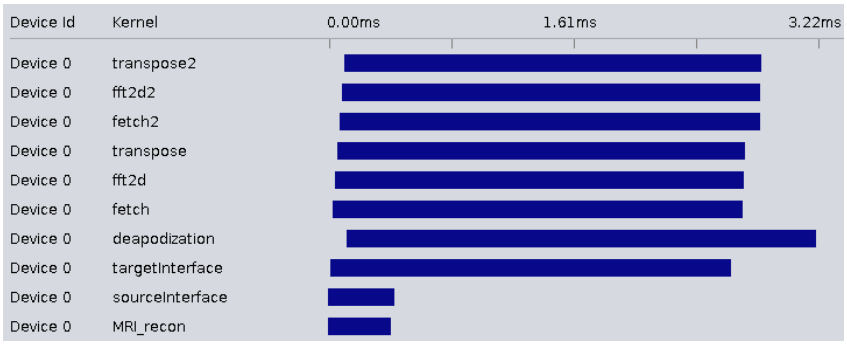


Figure 30: Altera OpenCL profiler output for modified FFT kernel

6.2 Half Fourier and Partial Echo

In chapter 2, these 2 techniques used to reduce the MRI acquisition time have been presented. These techniques exploit the symmetric behavior of the k-space to acquire just half of the data and to generates the other half. A similar approach can be used to reduce the number of processed points in the regridding process, reducing both the local memory required and the execution time.

Half Fourier filling consists in filling just half of the k-space in the phase encoding direction. The other half can be reproduced considering the symmetry of the k-space. This property can be exploited as well in the regridding step. In our work this technique was applied to the Stanford data set, considering first 90% of the phase encoding, then moving the boundary of 10% up to 50% of the phase encoding as shown in Figure 31.

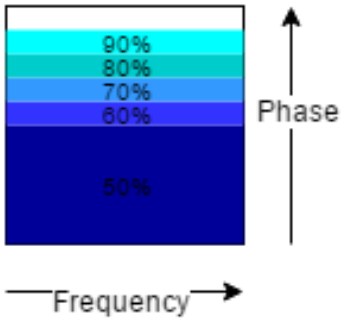


Figure 31: Partial k-space considered for the reconstruction, from 90% to 50% exploiting the Half Fourier Technique

The consideration of just a part of the k-space will result in a series of advantages and disadvantages:

- The number of processed source points is reduced, thereby improving the performance of the architecture.
- The size of the local buffer stored in target interface is reduced. A part of the target points is generated exploiting the symmetric behavior of the k-space.
- The accuracy of the reconstructed image is affected by the reduced number of source points used.

The following table will highlight the results in terms of accuracy of the reconstructed image of the Stanford and Cardiac data sets, as well as the reduced size of the local buffer, and the expected improvement in execution time.

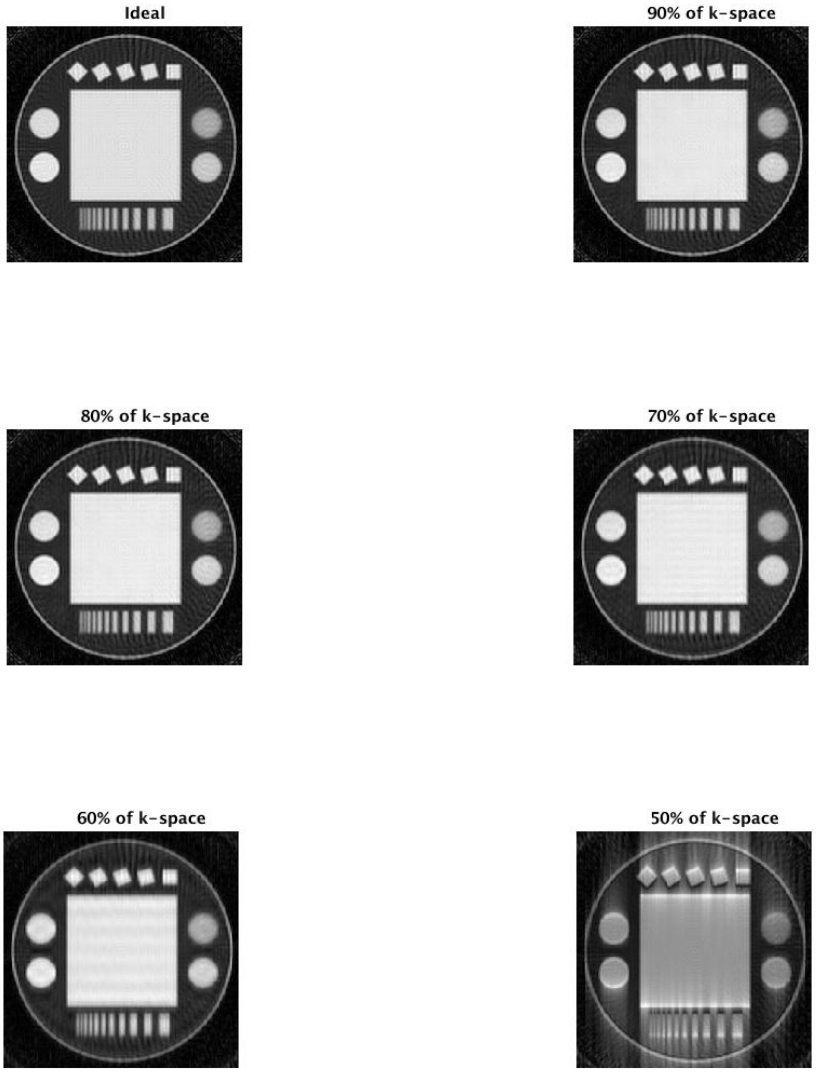


Figure 32: Reconstruction using part of source points sampled in the k-space for Stanford data set, for Spiral trajectory in Stanford Data set [28].

TABLE XIV: SSIM, PSNR, REDUCTION IN LOCAL BUFFER SIZE AND REDUCTION IN PROCESSED POINTS FOR REDUCED K-SPACE REGRIDDING, FOR SPIRAL TRAJECTORY FOR STANFORD DATA SET.

Used k-space percentage	SSIM	PSNR	Reduction in local buffer size	Reduction in processed point for Spiral trajecotry
90%	0.95	33.34	9%	4.44%
80%	0.94	31.43	19.36%	9.93%
70%	0.9	29.8	29.13%	19.47%
60%	0.85	26.97	39.29%	31.66%
50%	0.55	13.91	49.44%	50%

TABLE XV: SSIM, PSNR, REDUCTION IN LOCAL BUFFER SIZE AND REDUCTION IN PROCESSED POINTS FOR REDUCED K-SPACE REGRIDDING, FOR A RADIAL TRAJECTORY DATA SET.

Used k-space percentage	SSIM	PSNR	Reduction in local buffer size	Reduction in processed point for Radial trajecotry
90%	0.9	34.94	9%	2.86%
80%	0.88	33.24	19.36%	9.54%
70%	0.86	32.52	29.13%	17.94%
60%	0.78	26.76	39.29%	29.98%
50%	0.58	22.34	49.44%	50%

The same approach can be used in the Partial Echo reconstruction as in Figure 33. In this case just a part of the frequency encoding is considered, while the other part is derived from the properties of the k-space. Due to the symmetric behavior of the most used sampling trajectories the results obtained are practically the same both in performance of accuracy of the reconstructed image and on the gain in performance and memory utilization.

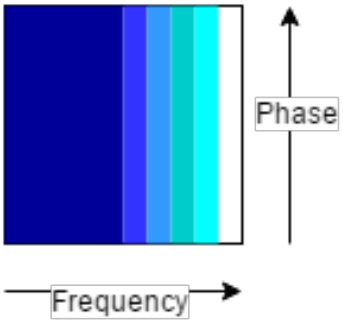


Figure 33: Partial k-space considered for the reconstruction, from 90% to 50% exploiting the Partial Echo Technique

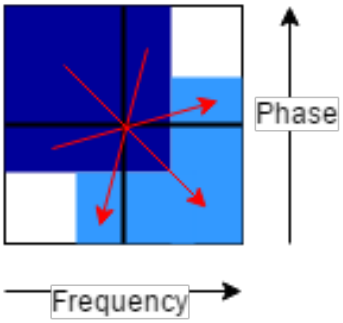


Figure 34: Combined Partial Reconstruction Example

These two techniques can be combined so to obtain a further improvement of performance and resource utilization further reducing the penalty in the accuracy. In this case source points below a certain limit both in the frequency and phase encoding are not considered. The reconstruction is then performed deriving some values from the symmetric properties of the k-space, and zero filling some part of it.

Figure 34 shows an example of the reconstruction procedure. The source points in the blue are properly utilized for the regridding. The grid points in the light blue are computed exploiting the k-space property. The grid locations in the white are filled with zeros. This shouldn't considerably affect the accuracy since this part of the k-space does not bring fundamental informations. As in the previous case, reconstruction data for Stanford and Cardiac data set are collected, and the actual image are proposed. This technique is able to further reduce the number of points processed and the size of the local buffer in the architecture, guaranteeing decent results in terms of accuracy.

6.3 Multiple Processing Blocks

Combining the fixed point implementation with the theoretical technique described above it could be possible to fit, in an FPGA, multiple hardware processing entities. This solution could be useful for real time MRI that is one of the trending topics in this field. From a simple estimation it could be possible to process up to 4 images of size 256x256 in parallel in an Altera Arria 10. The bottleneck is the amount of available local memory. Using the combined reconstruction technique proposed in the above section it is possible to reduce the amount of local used memory to 22% of the total availability for each instance. A concurrent design could also be useful for 3D MRI that requires a high number of frames to be processed.

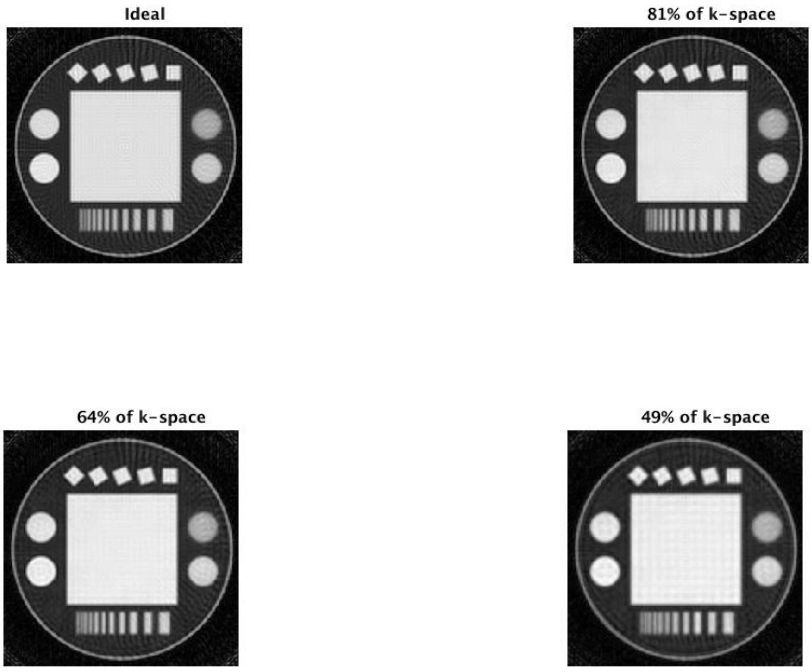


Figure 35: Reconstruction using part of source points sampled in the k-space for Stanford data set, for Spiral trajectory in Stanford Data set [28].

TABLE XVI: SSIM, PSNR, REDUCTION IN LOCAL BUFFER SIZE AND REDUCTION IN PROCESSED POINTS FOR COMBINED REDUCED K-SPACE REGRIDDING, FOR SPIRAL TRAJECTORY FOR STANFORD DATA SET.

Used k-space percentage	SSIM	PSNR	Reduction in local buffer size	Reduction in processed point for Radial trajecotry
81%	0.95	32.06	18.27%	6.91%
64%	0.89	28.56	35.28%	19.41%
49%	0.82	26.77	49.18%	33.30%

TABLE XVII: SSIM, PSNR, REDUCTION IN LOCAL BUFFER SIZE AND REDUCTION IN PROCESSED POINTS FOR COMBINED REDUCED K-SPACE REGRIDDING, FOR A RADIAL TRAJECTORY DATA SET.

Used k-space percentage	SSIM	PSNR	Reduction in local buffer size	Reduction in processed point for Spiral trajecotry
81%	0.91	35.89	18.27%	5.75%
64%	0.89	32.88	35.28%	16.80%
49%	0.87	32.77	49.18%	31.82%

CHAPTER 7

CONCLUSIONS

This document addresses the problem of image reconstruction for MRI applications using FPGAs. Recently non uniform sampling trajectories are being used in MRI to reduce the acquisition time consequently increasing the amount of mathematical operations needed to produce an accurate result. The Non uniform Fast Fourier Transform algorithm is able to reconstruct the image with an approximation scheme considerably reducing the computational time.

In this thesis an image reconstruction algorithm is, then, developed and implemented using OpenCL exploiting a powerful compiler from Altera. The design procedure is completely different with respect to any classical HDL, but, following the guidelines provided on Altera's website and decoupling the design from the actual hardware implementation, it is possible to obtain excellent results in a reasonable time. Different approaches for the density compensation step are analyzed and discussed in term of performance, accuracy of the results, and resource utilization in Arria 10. Then an efficient, low-power, and scalable architecture able to handle all the most common frame sizes in MRI is proposed. This architecture competes with and improves upon different applications provided in the literature, even if most of them just focus on the interpolation phase which is the most computational intensive.

The proposed implementation is then modified to improve performance. Altera OpenCL channel extension is used instead of a buffer in global memory. Although this solution, by far, improves the execution time, it loses the scalable nature. Fixed point arithmetic, which is not

directly supported by OpenCL, is used to describe the same implementation. This arithmetic led to a great reduction in terms of hardware utilization. Performance and accuracy are still not comparable with the floating point implementation. Furthermore, some parameters need to be properly adapted and changed with the MRI machine used for the acquisition.

Finally some further innovations to reduce the execution time are proposed. The first one is purely architectural and consists of a duplication of the FFT hardware so to avoid the Host interfering with the FPGA, while the second one studies the accuracy of the image reconstruction considering some properties of the k-space. The interpolation considers just a part of the source points, and the entire target grid space is estimated from that. The results show a great trade-off between size of the local buffer and estimated execution time with the accuracy of the reconstructed image.

APPENDICES

Appendix A

VORONOI DIAGRAM COMPUTATION ALGORITHMS

The Voronoi diagram consists of a partitioning of a space into regions depending on the distance with respect to given points. In the case of MRI acquisition these points are the source points, and the Voronoi diagram can be used in order to estimate the density weight for each source point. The computation of the Voronoi diagram is intensive and it is dependent on the number of points in the space, and on the wanted resolution.

The approach used in the Host code of the proposed architecture is slightly different but it is effective in the computation of the Voronoi area associated with each source point. A division of the source point space is assumed. The denser the "sampling" of the Voronoi diagram wanted, the more precise will be the result.

```
for each coordinate of the source point space K_{x,y}
        set minimum distance to 1000
        set closest source point to 0
        for each source points SP_i
                compute the distance between the source point SP_i and K_{x,y}
                if the distance is less that the minimum distance
                        set minimum distance to distance
                        set closest source point to SP_i
                end if
```

Appendix A (continued)

```
        end for
        increase the location SP_i of the weight vector
end for
```

This algorithm does not require any modification for the convex hull points. It is also possible to reduce the computation time of this algorithm exploiting the solution proposed in [18]. In that case, just the area of the polygons at the center of the k space can be computed, reducing the number of points to be assigned to a specific source point. Also the source point vector can be properly modified to consider only the source points at the center of the k space instead of all of them. This will considerably reduce the execution time of the algorithm, without losing in the reconstruction accuracy, as shown in the Accuracy results section.

Appendix B

FLOATING AND FIXED POINT ARITHMETIC

B.1 Floating Point

Altera OpenCL compiler describes floating point numbers using a technical standard provided by IEEE: IEEE 754-2008.

The format defined by the standard includes:

- **Finite Numbers**: each finite number is described by three integers:

$$(-1)^s \cdot c \cdot 2^q \tag{B.1}$$

  where s is the sign, c is the coefficient and q is the exponent.

- **Two infinites**

- **Two kind of** Nan

The possible representable values are determined by the number of digits of the coefficient, and the maximum value of the exponent parameter. OpenCL defines 3 types data types that can be used to describe floating point values. The difference is in the number of bit used to represent the number.

1. **cl_half**: it is a 16-bit float. 5 bits are used for the exponent, while 11 bits are used for the number representation.

Appendix B (continued)

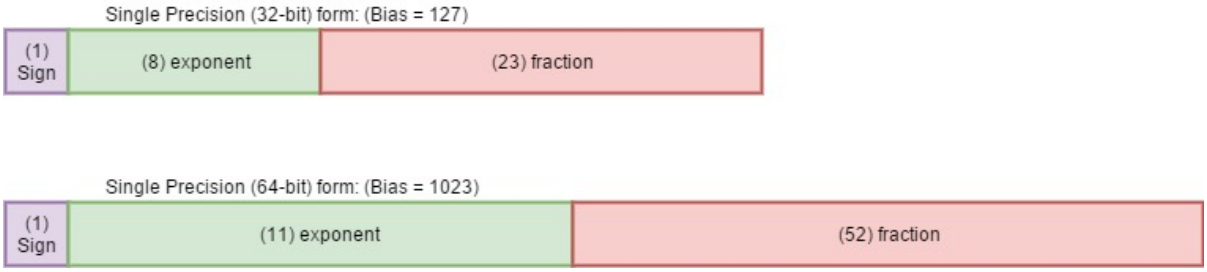


Figure 36: Floating Point Bit Division

2. **cl_float**: it is a 32-bit float. It represents a single precision float, 8 bits are used for the exponent, while 24 for the number representation.
3. **cl_double**: it is a 64-bit float. It represents a double precision float, 11 bits are used for the exponent, while 53 for the number representation.

The bits for the single and double representation are arranged as in Figure 36. The most significant bit is the sign, the biased exponent is in the middle while the mantissa i the least significant bits.

This representation is attractive to have great precision adding, though, some complexity in the management of the operations. For instance the multiplication between two floating point numbers requires different steps; a summation of the exponents of the two numbers, a multiplication of the mantissas, a normalization of that result, adjusting the exponent if necessary. Finally, a XOR between the two sign bits is necessary. A similar complex operation for the division has to be performed.

Appendix B (continued)

In an FPGA design this can lead to extra hardware in order to deal with all the operations, and, even if they are handled directly the compiler extra logic is require and can lead to a much complex hardware.

B.2 Fixed Point

"A fixed point number representation is a real data type for a number that has a fixed number of digits after the radix point" [32]. Essentially, a fixed point number is an integer scaled by a factor. Operations in fixed point arithmetic are simple and they can be easily implemented. To convert a fixed point number from a scaling factor F to another with factor G just a multiplication by F and a division by G is needed. In binary representation this is pretty easy because both multiplication and division can be implemented with a shifting operation (as long as the factor are power of 2).

Addition and subtraction of values of the same fixed point type is directly the mathematical operation between two integer, and the result is correct as long there is no overflow. The multiplication, as the addition and subtraction, is the simple multiplication between two integers. It is, though, necessary to take into account the fact that the result will have a scaling factor that is the product of the scaling factors of the two operands. A similar process has to be applied also for the division, where the scaling factor is given gy the quotient by the 2 scaling factors.

The problem with fixed point data type is that a lot of operations can produce results that have more bits than the operands, so information loss is probable. This has to be taken into

Appendix B (continued)

account when designing a fixed point architecture.

Because of that, the hardware necessary to implement fixed point operations is smaller and less complex than the equivalent implemented in floating point arithmetic. The result is that is possible to fit more fixed point operations into the FPGA. This can lead to better performance, if there is the chance to double the architecture within the FPGA allowing to double the throughput, or to free space in the local memory so to fit bigger buffers in the Target Interface kernel.

CITED LITERATURE

1. Pezzotti, E., Iacobucci, A., Nash, G., Cheema, U., Vinella, P., and Ansari, R.: Fpga-based hardware accelerator for image reconstruction in magnetic resonance imaging (abstract only). In Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA '17, pages 293–293, New York, NY, USA, 2017. ACM.

2. Iacobucci, A.: Architecture Design for Efficient Non-uniform Fast FourierTransform (NuFFT) Implementation of FPGA. Master's thesis, UIC, 2017.

3. Elmaoğlu, M. and Çelik, A.: Fundamentals of Magnetic Resonance Imaging, pages 7–23. Boston, MA, Springer US, 2012.

4. Zeng, G. L.: MRI Reconstruction, pages 175–192. Berlin, Heidelberg, Springer Berlin Heidelberg, 2010.

5. Mezrich, R.: A perspective on k-space. Radiology, 195(2):297–315, 1995. PMID: 7724743.

6. Pratt, W. K.: Unitary Transforms, pages 189–216. John Wiley & Sons, Inc., 2006.

7. Potts, D., Steidl, G., and Tasche, M.: Fast Fourier Transforms for Nonequispaced Data: A Tutorial, pages 247–270. Boston, MA, Birkhäuser Boston, 2001.

8. Kestur, S., Park, S., Irick, K. M., and Narayanan, V.: Accelerating the nonuniform fast fourier transform using fpgas. In Field-Programmable Custom Computing Machines (FCCM), 2010 18th IEEE Annual International Symposium on, pages 19–26, May 2010.

9. Schomberg, H. and Timmer, J.: The gridding method for image reconstruction by fourier transformation. IEEE Transactions on Medical Imaging, 14(3):596–607, Sep 1995.

10. Pauly, J.: Non-cartesian reconstruction. http://mri-q.com/uploads/3/4/5/7/34572113/pauly-non-cartesian_recon.pdf, 2005.

11. Meyer, C. H., Hu, B. S., Nishimura, D. G., and Macovski, A.: Fast spiral coronary artery imaging. Magnetic Resonance in Medicine, 28(2):202–213, 1992.

CITED LITERATURE (continued)

12. Heid, O.: Archimedian spirals with euclidean gradient limits. In Proc. ISMRM, 4th Annu. Meeting, page 114, 1996.

13. Spielman, D. M., Pauly, J. M., and Meyer, C. H.: Magnetic resonance fluoroscopy using spirals with variable sampling densities. Magnetic resonance in medicine, 34(3):388–394, 1995.

14. Jackson, J. I., Meyer, C. H., Nishimura, D. G., and Macovski, A.: Selection of a convolution function for fourier inversion using gridding [computerised tomography application]. IEEE transactions on medical imaging, 10(3):473–478, 1991.

15. Liao, J. and Pelc, N.: Image reconstruction of generalized spiral trajectory. In Proc. ISMRM, 4th Annu. Meeting, page 359, 1996.

16. Zhao, Y., Liu, S., and Zhang, Y.: Spatial density voronoi diagram and construction. Journal of Computers, 7(8):2007–2014, 2012.

17. Rasche, V., Proksa, R., Sinkus, R., Bornert, P., and Eggers, H.: Resampling of data between arbitrary grids using convolution interpolation. IEEE transactions on medical imaging, 18(5):385–392, 1999.

18. Malik, W. Q., Khan, H. A., Edwards, D. J., and Stevens, C. J.: A gridding algorithm for efficient density compensation of arbitrarily sampled fourier-domain data. In IEEE/Sarnoff Symposium on Advances in Wired and Wireless Communication, 2005., pages 125–128. IEEE, 2005.

19. Pellerin, D. and Thibault, S.: Practical fpga programming in c. Prentice Hall Press, 2005.

20. Czajkowski, T. S., Neto, D., Kinsner, M., Aydonat, U., Wong, J., Denisenko, D., Yiannacouras, P., Freeman, J., Singh, D. P., and Brown, S. D.: Opencl for fpgas: Prototyping a compiler. In Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA), page 1. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2012.

21. Singh, D. and Engineer, S. P.: Higher level programming abstractions for fpgas using opencl. In Workshop on Design Methods and Tools for FPGA-Based Acceleration of Scientific Computing, 2011.

CITED LITERATURE (continued)

22. Cheema, U. I., Nash, G., Ansari, R., and Khokhar, A. A.: Power-efficient re-gridding architecture for accelerating non-uniform fast fourier transform. In 2014 24th International Conference on Field Programmable Logic and Applications (FPL), pages 1–6. IEEE, 2014.

23. Andrade, J., da Silva, V. M. M., and Fernandes, G. F. P.: From opencl to gates: The fft. In GlobalSIP, pages 1238–1241, 2013.

24. Beatty, P. J., Nishimura, D. G., and Pauly, J. M.: Rapid gridding reconstruction with a minimal oversampling ratio. IEEE transactions on medical imaging, 24(6):799–808, 2005.

25. Huynh-Thu, Q. and Ghanbari, M.: Scope of validity of psnr in image/video quality assessment. Electronics letters, 44(13):800–801, 2008.

26. Wang, Z., Bovik, A. C., Sheikh, H. R., and Simoncelli, E. P.: Image quality assessment: from error visibility to structural similarity. IEEE transactions on image processing, 13(4):600–612, 2004.

27. Hore, A. and Ziou, D.: Image quality metrics: Psnr vs. ssim. In Pattern recognition (icpr), 2010 20th international conference on, pages 2366–2369. IEEE, 2010.

28. Pauly, J.: Standford ee369c: Medical image reconstruction. http://web.standford.edu/class/ee369c/data/vardens.mat, 2012.

29. Kestur, S., Park, S., Irick, K. M., and Narayanan, V.: Accelerating the nonuniform fast fourier transform using fpgas. In Field-Programmable Custom Computing Machines (FCCM), 2010 18th IEEE Annual International Symposium on, pages 19–26. IEEE, 2010.

30. Sørensen, T. S., Schaeffter, T., Noe, K. Ø., and Hansen, M. S.: Accelerating the nonequispaced fast fourier transform on commodity graphics hardware. IEEE Transactions on Medical Imaging, 27(4):538–547, 2008.

31. Li, L. and Wyrwicz, A. M.: Design of an mr image processing module on an fpga chip. Journal of Magnetic Resonance, 255:51–58, 2015.

32. Nambiar, V. P., Khalil-Hani, M., Sahnoun, R., and Marsono, M.: Hardware implementation of evolvable block-based neural networks utilizing a cost efficient sigmoid-like activation function. Neurocomputing, 140:228 – 241, 2014.

CITED LITERATURE (continued)

33. Weiger, S., Oberhammer, T., and Hennel, F.: Design of a gradient waveform for a k-space trajectory with an upper frequency limit due to mri gradient hardware, October 5 2011. EP Patent App. EP20,100,157,581.

VITA

NAME	Emanuele Pezzotti
EDUCATION	
	Master of Science in "Electrical and Computer Engineering", University of Illinois at Chicago, Dec 2016, USA
	Specialization Degree in "Embedded Systems Electronic Engineering", Dec 2016, Polytechnic of Turin, Italy
	Bachelor's Degree in "Electronic Engineering", Oct 2014, Polytechnic of Turin, Italy
LANGUAGE SKILLS	
Italian	Native speaker
English	Full working proficiency
	2014 - TOEFL examination (99/110)
	A.Y. 2015/16 One Year and a half of study abroad in Chicago, Illinois
	A.Y. 2014/15. Lessons and exams attended exclusively in English
SCHOLARSHIPS	
Summer 2016	Research Assistantship (RA) position (20 hours/week) with full tuition waiver plus monthly stipend
Spring 2016	Teaching Assistantship (TA) position (10 hours/week) with full tuition waiver plus monthly stipend
Fall 2015	Italian scholarship for final project (thesis) at UIC
Fall 2015	Italian scholarship for TOP-UIC students
WORK EXPERIENCE AND PROJECTS	
Sep 2016 - Dec 2016	Embedded Development Intern at Swipesense
Gen 2016 - Dec 2016	Collaboration with Altera for accelerator design on FPGA using OpenCL
2015	Top-down design of a custom DLX processor 2015

VITA (continued)

	Design of digital micro and macro-architectures: design, VHDL description, simulation, synthesis, place & route of DLX microprocessor in all its part starting from scratch, pipelined version, data and control hazard management.
2014-2015	Other Experiences:
	Use of prototyping boards, micro controllers, FPGAs, inside a bigger system or stand-alone (experience: realization of some measuring and control systems, simulations of designed components on FPGA)
	Development of all the parts of the software needed to realize measuring systems, data processing systems or mechanical control
	Scripts for simulation and synthesis of integrated circuits
	Multi-process and multi-thread C programs